**Noname manuscript No.**
(will be inserted by the editor)

# Techniques for Gradient Based Bilevel Optimization with Nonsmooth Lower Level Problems

**Peter Ochs · René Ranftl · Thomas Brox · Thomas Pock**

Received: date / Accepted: date

**Abstract** We propose techniques for approximating bilevel optimization problems with non-smooth and non-unique lower level problems. The key is the substitution of the lower level minimization problem with an iterative algorithm that is guaranteed to converge to a minimizer of the problem. Using suitable non-linear proximal distance functions, the update mappings of such an iterative algorithm can be differentiable, notwithstanding the fact that the minimization problem is nonsmooth. This technique for smoothly approximating the solution map of the lower level problem raises several questions that are discussed in this paper.

P. Ochs
Mathematical Image Analysis Group
University of Saarland, Germany
E-mail: ochs@mia.uni-saarland.de

R. Ranftl
Visual Computing Lab
Intel Labs, Santa Clara, CA, United States
E-mail: rene.ranftl@intel.com

T. Brox and P. Ochs
Computer Vision Group
University of Freiburg, Germany
E-mail: {ochs,brox}@cs.uni-freiburg.de

T. Pock
Institute for Computer Graphics and Vision
Graz University of Technology, Austria
and
Digital Safety & Security Department
AIT Austrian Institute of Technology GmbH
1220 Vienna, Austria
E-mail: pock@icg.tugraz.at

## 1 Introduction

We consider numerical methods for solving bilevel optimization problems of the form

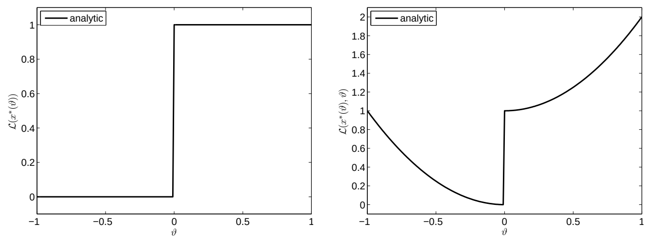
$$\begin{aligned} &\min_{\vartheta}\ \mathcal{L}(x^*(\vartheta),\vartheta) \\ &s.t.\ x^*(\vartheta) \in \arg\min_{x\in\mathbb{R}^N} E(x,\vartheta)\,, \end{aligned} \tag{1}$$

where $\mathcal{L}$ is a function penalizing the differences between the output of the lower level problem $x^*(\vartheta)$ and some given ground truth data. In addition, $\mathcal{L}$ can also contain a regularizer on the parameter vector $\vartheta$, e.g. a sparsity prior. The mapping $x^*(\vartheta)$ is the solution of an optimization problem (parametrized by $\vartheta$).

In the context of game theory and often also in the general bilevel literature, the bilevel optimization problem is represented as a leader–follower problem. The leader (upper level problem) tries to optimize the next move (minimization of the upper level problem) under consideration of the move of an opponent, the follower. Given some information $\vartheta$ to the follower, the leader tries to anticipate the follower's next move (minimization of the lower level problem).

In the context of machine learning or computer vision, bilevel problems can be seen as a formalization of parameter learning problems. The objective of the upper level problem $\mathcal{L}$ is usually denoted loss function. It invokes some prior assumptions to the parameter $\vartheta$ and measures the discrepancy between the solution $x^*(\vartheta)$ of an energy minimization problem for a given parameter $\vartheta$ and some ground truth. The lower level problem corresponds to an energy model that solves a specific task, e.g. multi-label segmentation.

In this paper, we focus on a class of problems that allows for non-smooth, convex functions $x \mapsto E(x,\vartheta)$ in the lower level problem, e.g. sparse models based on



(a) Loss functions for (2). (b) Loss functions for (3).

**Fig. 1** Problem in bilevel optimization problems containing non-smooth lower level problems with multiple solutions.

the $\ell_1$-norm. Such models have become very popular in the computer vision, image processing and machine learning communities since they are robust with respect to noise and outliers in the input data.

Due to the possibly high dimensionality of the parameter vector, we pursue the minimization of the bilevel problem (1) using gradient based methods. Hence, a descent direction of $\mathcal{L}$ with respect to $\vartheta$ must be determined. Its estimation involves the Jacobian of the solution map $x^*(\vartheta)$ with respect to the parameter vector $\vartheta$, which causes three kind of problems:

(i) The solution mapping $x^*(\vartheta)$ is defined implicitly (as a minimizer of the lower level problem).
(ii) The solution of the lower level problem might have multiple solutions.[1]
(iii) The lower level problem can be non-smooth and hence, derivatives cannot be computed in the original sense of smooth functions.

(i) In some cases, there is an explicit solution to the lower problem (depending on the parameter $\vartheta$). Then (1) reduces to a single level problem and the total derivative $d\mathcal{L}/d\vartheta$ can be obtained by the standard chain rule (assuming sufficient smoothness). However, this is rarely possible. If there is no explicit solution, but the lower level objective function is sufficiently smooth in both arguments and the problem is unconstrained, the lower level problem can be replaced by its optimality condition $\nabla_x E(x,\vartheta) = 0$ with respect to $x$. Under suitable conditions, the implicit function theorem (cf. Section 5.1) provides an explicit formula for the derivative of $x^*(\vartheta)$ with respect to $\vartheta$. However, this approach is limited since most popular models are non-smooth, and by smoothing these models often lose their nice properties.

(ii) Let us briefly discuss the problems arising from a lower level problem with non-unique solution. Consider the bilevel problem

$$\begin{aligned} &\min_{\vartheta\in\mathbb{R}}\ (x^*(\vartheta)-1)^2 \\ &s.t.\ x^*(\vartheta) \in \arg\min_{x\in[0,1]} \vartheta x\,, \end{aligned} \tag{2}$$

with lower level solution

$$x^*(\vartheta) = \begin{cases} 0, & \text{if } \vartheta > 0\,; \\ 1, & \text{if } \vartheta < 0\,; \\ [0,1], & \text{if } \vartheta = 0\,; \end{cases}$$

and

$$\mathcal{L}(x^*(\vartheta)) = \begin{cases} 1, & \text{if } \vartheta > 0\,; \\ 0, & \text{if } \vartheta < 0\,; \\ [0,1], & \text{if } \vartheta = 0\,. \end{cases}$$

The loss function is shown in Figure 1(a). The derivative of the loss function with respect to $\vartheta$ vanishes for all $\vartheta \neq 0$. Therefore a gradient based method will get stuck almost everywhere. Similar situations will arise also in high dimensional problems such as multi-label segmentation. Since small changes in the input data (pixel likelihoods) do not change the segmentation result, the energy landscape of the loss function will have the form of a high dimensional step function. Any gradient based method will be incapable of determining a descent direction. This problem can be relaxed to some extend by averaging many training examples but the principal problem will not disappear. A similar situation is to be expected for robust models that, by definition, are not affected by small perturbations of the input data (or the parameter $\vartheta$).

Another issue of a non-unique solution map $x^*(\vartheta)$ is the non-existence of a solution for (1), which the following example demonstrates:

$$\begin{aligned} &\min_{\vartheta\in\mathbb{R}}\ (x^*(\vartheta)-1)^2 + \vartheta^2 \\ &s.t.\ x^*(\vartheta) \in \arg\min_{x\in[0,1]} \vartheta x\,. \end{aligned} \tag{3}$$

The loss function is shown in Figure 1(b). In terms of leader–follower, the problem becomes obvious when the leader picks the wrong response from the follower. Assume for $\vartheta = 0$, the leader anticipates $x^*(\vartheta) = \frac{1}{2} \in [0,1]$. Then, the solution of the leader's problem does not exist, because the leader's optimization problem is not lower semi-continuous at 0, and values of $\vartheta$ approaching 0 from the left decrease the value of $\mathcal{L}$ towards 0, but $\mathcal{L}(\frac{1}{2},0) > 0$.

(iii) Due to the non-smoothness of the lower level problem, standard calculus cannot be applied. In variational (non-smooth) analysis, there are many generalizations of derivatives, such as the convex subdifferential, the Fréchet subdifferential, or the limiting subdifferential, etc., and generalizations of the chain rule rely

[1] Note that the bilevel problem as in (1) is not well-defined in this case. We discuss some details in Section 4.

on constraint qualifications that are sometimes quite restrictive and often hard to verify.

In the conference version of this paper [29], we introduced an approach to overcome the smoothness restriction in some cases of practical interest. The idea is to replace the lower level problem by an iterative algorithm that is guaranteed to converge to a solution of the problem. If the update-mapping of the algorithm is a smooth functions, the chain rule can be applied to the composition of these update-mappings recursively and the exact derivatives with respect to the parameter vector $\vartheta$ can be computed. Algorithms based on Bregman distances are key for this development. The number of iterations of the iterative algorithm steers the approximation quality to the lower level problem. We are going to argue that the number of iterations can be seen as a natural "smoothing parameter".

The convergence analysis of the derivative of the algorithm mapping for an increasing number of iterations is quite challenging and mainly a theoretical issue. It is left for future work. Thus, strictly speaking the proposed approach is a heuristic.

From a practical point of view, most iterative algorithms are stopped after performing a small number of iterations. Once the number of iterations is fixed, the resulting bilevel optimization problem—the lower level problem is replaced by the algorithm—seeks for an optimal $\vartheta$ for exactly the chosen algorithm with the fixed number of iterations. Thus, the descent direction is based on the derivative of the algorithm for which $\vartheta$ is to be optimized. This is in contrast to an approach based on the optimality condition of a smooth approximation to the lower level problem. The descent direction is based on the derivative of the optimality condition evaluated at the minimum of the lower level problem, which can only approximately be determined. In the end, the objective of the lower level problem (with optimized parameter $\vartheta$) is solved with a smaller number of iterations in practice. Hence, there are two natural sources of errors.

Beyond the analysis of the conference paper, we discuss approximations to the derivative evaluation that reduce the memory requirements and the computational cost significantly. We give some more details about the general implementation of our approach. Moreover, we consider the limiting case, i.e., the fixed point equation of an iterative algorithm in the lower level problem.

We point out several applications of our approach and evaluate it for a multi-label segmentation problem that is coupled with a convolutional neural network.

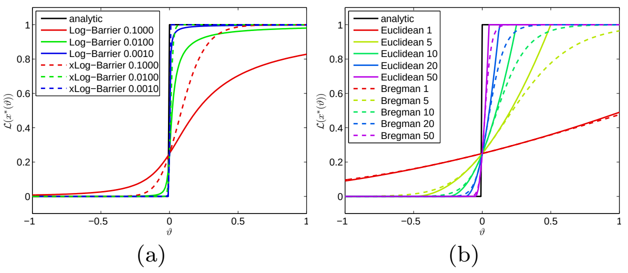


**Fig. 2** Approximations to (2) and (3) by adding a log barrier to the lower level problem or by approximating the lower level problem with an iterative algorithm.

## 2 Outline

In the related work section, we arrange our approach in the general bilevel optimization literature, point out connection to structured support vector machines, and discuss applications in computer vision and machine learning. Section 4 presents the details of the general bilevel optimization problem that is addressed in this paper. Section 5 is focused on the estimation of a descent direction that can be used in gradient based algorithms for solving the bilevel optimization problem. The approach in Section 5.1 requires a smooth approximation of the lower level problem. To solve the problem in (2), the technique in Section 5.1 can be applied, for instance, to

$$x^*(\vartheta) \in \arg\min_{x\in\mathbb{R}} \; \vartheta x - \mu(\log(x) + \log(1-x))$$

or

$$x^*(\vartheta) \in \arg\min_{x\in\mathbb{R}} \; \vartheta x + \mu(x\log(x) + (1-x)\log(1-x)) \,,$$

where $\mu > 0$ regulates the approximation quality of the log-barriers. In Figure 2(a) the first is denoted `Log-Barrier` and the latter `xLog-Barrier`. Since, both approximations are sufficiently smooth in $(-1,1)$, derivatives can be estimated using the implicit function theorem. In contrast, our approach in Section 5.2 approximates the lower level problem by forward–backward splitting with Bregman proximity functions. In Figure 2(b) `Bregman` shows this smooth approximation controlled by the number of iterations. In the limit, our smooth approximation solves a non-smooth optimization problem. `Euclidean` in Figure 2(b) demonstrates that a Euclidean proximity function results in a non-smooth approximation of the lower level problem. Finally, in Section 5.3, we consider the fixed point equation of an algorithm as a substitute of the lower level problem.

In order to simplify the implementation, the ideas from Section 5 are specialized to the forward–backward splitting algorithm and the Chambolle–Pock primal–dual algorithm, both involving Bregman proximity functions. Section 7 discusses details, examples, and applications of Bregman proximity functions and of the algorithms introduced in Section 5.

The three approaches from Section 5 are analyzed in a toy example in Section 8. In Section 9 a large scale optimization problem with a multi-label segmentation model parameterized by a convolutional neural network is presented.

## 3 Related Work

We propose a simple approximation of the lower level problem that naturally addresses *non-smoothness* and *non-uniqueness*.

For a non-unique solution map (a set-valued mapping) of the lower level problem (1) is not even well-defined (cf. Remark 1). [13] describes three possible options to cope with this problem. The *optimistic bilevel optimization problem* assumes a cooperative strategy of leader and follower, i.e., in case of multiple solutions the follower tries to minimize the upper level objective. The *pessimistic bilevel problem* is the other extreme problem, where the leader tries to bound the damage that the follower could cause by his move. The *selection function approach* assumes that the leader can always predict the followers choice. Of course, these three approaches are the same for lower level problems with a unique output.

Our approach does not fall into any of the three cases, however the selection function approach is the closest. The difference is that our approximation changes the output also at (originally) unique points. Our solution strategy reduces the solution map to be single-valued, alike the approaches mentioned above.

[13] classifies the following sought optimality conditions[2]. The *primal Karush–Kuhn–Tucker (KKT) transformation* replaces the lower level problem by the necessary (and sufficient) optimality condition for a convex function. The equivalence to the original problem is shown in [14]. The *classical KKT transformation* substitutes the lower level problem with the classical KKT conditions. As a consequence of an extra variable, the problems are not fully equivalent anymore (see [13]). This approach, which leads to a non-smooth mathematical problem with complementary constraints (MPEC), is the most frequently used one. The third approach is the *optimal value transform*, which introduces a constraint that bounds the lower level objective by the optimal value function.

[2] The classification in [13] applies to the optimistic bilevel problem.

Our approach is—in the limit—motivated by the first class of the primal KKT transformation. We consider the fixed point equation of an algorithm, which represents the optimality condition (without introducing additional variables), and approximate this situation with finitely many iterations of the algorithm. The classical KKT conditions are, for example, used in [9, 10, 22, 34].

As pointed out in the introduction, we focus on gradient based methods, such as gradient descent, L-BFGS [24], non-linear conjugate gradient [18, 1], Heavy-ball method [39], iPiano [28], and others, for solving the bilevel optimization problem. In particular, this paper focuses on the estimation of descent directions. One option is to numerically approximate the gradient with finite differences like in [15]. However, we pursue what is known as algorithmic/automatic differentiation. It is based on the idea to decompose the derivative evaluation into small parts by means of a chain rule, where the analytic derivative of each part is known. A whole branch of research deals with this technique [20]. Obviously, the idea to differentiate an algorithm in the lower level problem is not new [36, 16]. The difference is that our algorithm has a smooth update rule while actually minimizing a non-smooth objective function. Another idea to approach a non-smooth problem with an iterative algorithm is presented in [11], where a chain rule for weak derivatives is used (cf. Section 5.4).

In machine learning, parameter learning has a long tradition. While classical approaches are based on maximum likelihood learning, more recent approaches are based on the Structured Output Support Vector Machine [37]. Consider again the bilevel optimization problem (1), where we assume that the function $\mathcal{L}(x^*(\vartheta),\vartheta)$ is given by

$$\mathcal{L}(x,\vartheta) = \varDelta(x,y) + R(\vartheta)\,,$$

where $\varDelta(x^*(\vartheta), y)$ is a loss function that penalizes errors with respect to the ground truth solution $y$ and $R(\vartheta)$ is a convex regularizer on the parameter vector. Furthermore, we assume that the parameter vector $\vartheta$ depends linearly on the lower level energy, such that $E(x,\vartheta)$ can be written as $E(x,\vartheta) = \sum_{i=1}^n \vartheta_i e_i(x)$. Observe that this structure is common to many popular models in machine learning and computer vision.

Now, the idea is to replace the upper level objective function by an upper bound. Let $x^*$ be a minimizer of the lower level problem, then for all vectors $\bar{x}$ one has:

$$\begin{aligned}\Delta(x^*, y) &\leq \Delta(x^*, y) + E(\bar{x}, \vartheta) - E(x^*, \vartheta) \\ &\leq \max_x \Delta(x, y) + E(\bar{x}, \vartheta) - E(x, \vartheta)\,.\end{aligned}$$

Observe that the right hand side is convex in $\vartheta$, since it is a maximum over linear functions. Hence, the overall parameter learning problem has become a convex problem. Here, $\bar{x}$ is an arbitrary solution vector which however should be chosen as close as possible to a minimizer $x^*$ of the lower level problem. Often the choice $\bar{x} = y$ is made since one would like to force the energy of the lower level problem to be at least as good as the energy of the ground truth solution. Also observe that in case $x^* = y$, i.e. the lower level problem can reproduce the ground truth solution exactly, the convex upper bound is exact. In case this is not possible (which is the usual case) the convex upper bound only provides an approximation to the problem.

The resulting parameter learning problem, can be solved using a subgradient descent algorithm. In order to compute subgradients one needs to solve problems of the form

$$\max_x \Delta(x, y) - E(x, \vartheta) = -\min_x E(x, \vartheta) - \Delta(x, y).$$

Hence, it is advisable to pick a loss function $\Delta(x, y)$ that still allows to compute exact solutions of the lower level problem.

Examples for applications of bilevel optimization in the computer vision and machine learning community are the following. Bilevel optimization was considered for task specific sparse analysis prior learning [31] and applied to signal restoration. In [22,9,10] a bilevel approach was used to learn a model of natural image statistics, which was then applied to various image restoration tasks. A variational formulation for learning a good noise model was addressed in [34] in a PDE-constrained optimization framework, with some follow-up works [5,33,6]. In machine learning bilevel optimization was used to train a SVM [3] and other techniques [26]. Recently, it was used for the end-to-end training of a Convolutional Neural Network (CNN) and a graphical model for binary image segmentation [32] (cf. Section 9).

Finally, we want to refer to [12] for an annotated bibliography with many references regarding the theoretical and practical development in bilevel optimization.

**Preliminaries**

If not stated different, we will always work in a Euclidean vector space $\mathbb{R}^N$ of dimension $N$ equipped with the standard Euclidean norm $\|\cdot\| := \sqrt{\langle\cdot,\cdot\rangle}$ that is induced by the standard inner product. We use the notation $\overline{\mathbb{R}} := \mathbb{R} \cup \{\infty\}$ to denote the extended real numbers.

We use the notation $[x * a]$ for $x, a \in \mathbb{R}^N$ to denote the set $\{x \in \mathbb{R}^N \,|\, \forall i\colon\ x_i * a_i\}$, where $* \in \{<, \leq, =, \geq, >\}$ is a binary relation on $\mathbb{R} \times \mathbb{R}$. For example $[x \geq 0]$ denotes the non-negative orthant in $\mathbb{R}^N$.

## 4 The Bilevel Problem

The bilevel optimization problem considered in this paper is the following:

$$\begin{aligned}&\min_{\vartheta \in \mathbb{R}^P} \ \mathcal{L}(x^*(\vartheta), \vartheta) + \ell(\vartheta) \\ &s.t.\ x^*(\vartheta) \in \arg\min_{x \in \mathbb{R}^N} E(x, \vartheta)\end{aligned} \tag{4}$$

The function $\ell\colon \mathbb{R}^P \to \overline{\mathbb{R}}$ is assumed to be proper, lower semi-continuous, convex, and "prox-friendly"[3] and the function $\mathcal{L}\colon \mathbb{R}^N \times \mathbb{R}^P \to \mathbb{R}$ to be continuously differentiable on $\operatorname{dom} \ell$. The optimization variable is the (parameter) vector $\vartheta \in \mathbb{R}^P$. It appears implicit and explicit in the upper level problem. It is implicit via the solution mapping $x^*(\vartheta) \in \mathbb{R}^N$ of the lower level problem and explicit in $\ell$ and in the second argument of $\mathcal{L}$. The lower level is a minimization problem in the first variable of a proper, lower semi-continuous function $E\colon \mathbb{R}^N \times \mathbb{R}^P \to \overline{\mathbb{R}}$. For each $\vartheta \in \mathbb{R}^P$ the objective function (energy) $x \mapsto E(x, \vartheta)$ is assumed to be convex.

Note that our formulation includes constrained optimization problems in the upper and lower level problem. The functions $\ell$ and $E$ are defined as extended-valued (real) functions. Of course, in order to handle the constraints efficiently in the algorithm, the constraint sets should not be too complicated.

In order to solve (4), we can apply iPiano [28], a gradient-based algorithm that can handle the non-smooth part $\ell(\vartheta)$. The extension of iPiano in [27, Chapter 6] allows for a prox-bounded (non-convex, non-smooth) function $\ell(\vartheta)$. Informally, the update step of this algorithm (for the parameter vector $\vartheta$) reads

$$\begin{aligned}\vartheta^{k+1} \in \operatorname{prox}_{\alpha_k \ell} \big(\vartheta^k - \alpha_k \nabla_\vartheta \mathcal{L}(x^*(\vartheta^k), \vartheta^k) \\ + \beta_k(\vartheta^k - \vartheta^{k-1})\big)\,,\end{aligned} \tag{5}$$

[3] The associated proximity operator has a closed-form solution or the solution may be determined efficiently numerically.

where $\text{prox}_{\alpha_k \ell}$ denotes the proximity operator of the function $\ell$, and $\alpha_k$ is a step-size parameter and $\beta_k$ steers the so-called inertial effect of the algorithm (usually $\beta_k \in [0,1]$). For details about $\alpha_k$ and $\beta_k$, we refer to [28,27], where convergence to a stationary point (a zero in the limiting subdifferntial) is proved under mild assumptions. If the non-smooth term is not present, several gradient based solvers can be used, e.g. [24,39,18,1].

(5) points out the main aspect in applying such a gradient-based algorithm, namely the evaluation of the gradient $\nabla_\vartheta \mathcal{L}(x^*(\vartheta),\vartheta)$. The remainder of this paper deals with exactly this problem: computing $\nabla_\vartheta \mathcal{L}(x^*(\vartheta),\vartheta)$ with a solution mapping $\vartheta \mapsto x^*(\vartheta)$ of a possibly non-smooth objective function in the lower level.

*Example 1* A simple example of practical interest in image processing is the following image denoising problem

$$E(x,\vartheta) = \frac{1}{2}\|x-b\|_2^2 + \vartheta\|\mathcal{D}x\|_1\,,$$

where $b$ is a noisy vector (that represents an image) and $\mathcal{D}x$ computes partial derivatives of the image domain. Therefore $\|\mathcal{D}x\|_1$ penalizes variations in the result (image) $x$.

It can be used in the lower level of a bilevel optimization problem whose goal is to find the parameter $\vartheta \geq 0$ that best reconstructs the original image $\bar{b}$ by solving the denoising problem $\min_x\ E(x,\vartheta)$. A suitable upper level objective function is

$$\mathcal{L}(x^*(\vartheta),\vartheta) + \ell(\vartheta) = \frac{1}{2}\|x^*(\vartheta) - \bar{b}\|^2 + \delta_{[\vartheta\geq 0]}(\vartheta)\,.$$

*Remark 1* The formulation (4) of a bilevel optimization problem only makes sense when $\arg\min_{x\in\mathbb{R}^N} E(x,\vartheta)$ yields a unique minimizer. In that case optimality of the bilevel problem can be derived from standard optimality conditions in non-linear programming. If the lower level problem does not provide a unique solution, the loss function $\mathcal{L}$ actually needs to be defined on the power set of $\mathbb{R}^n$ and a different notion of optimality needs to be introduced. Since, this results in problems that are not relevant for this paper, we refer the interested reader to [13]. A common circumvention is to consider the corresponding optimistic bilevel problem.

## 5 Computing descent directions

For a given parameter value $\vartheta \in \mathbb{R}^P$, we would like to compute a descent direction of $\mathcal{L}$ in (4) with respect to $\vartheta$ in order to find a numerical solution using some gradient based method. Obviously, we need the derivative of the solution map $x^*(\vartheta)$ with respect to $\vartheta$. In the following, we present strategies to approximate the (possibly non-smooth) lower level problem and to compute a descent direction.

### 5.1 Derivative of a smoothed lower level problem

If we assume that the objective function of the lower level problem of (4) is twice continuously differentiable, we can make use of the implicit function theorem to find the derivative of the solution map with respect to $\vartheta$. The optimality condition of the lower level problem is $\nabla_x E(x,\vartheta) = 0$, which under some conditions implicitly defines a function $x^*(\vartheta)$. Let us define for a moment $F(x,\vartheta) = \nabla_x E(x,\vartheta)$. As we assume that the problem $\min_x E(x,\vartheta)$ has a solution, there is $(x^*,\bar{\vartheta})$ such that $F(x^*,\bar{\vartheta}) = 0$. Then, under the conditions that $F$ is continuously differentiable and $(\partial F/\partial x)(x^*,\bar{\vartheta})$ is invertible, there exists an explicit function $x^*(\vartheta)$ defined in a (open) neighborhood of $x^*$. Moreover, the function $x^*(\vartheta)$ is continuously differentiable at $\bar{\vartheta}$ and it holds that

$$\frac{\partial x^*}{\partial \vartheta}(\bar{\vartheta}) = \left(-\frac{\partial F}{\partial x}(x^*(\bar{\vartheta}),\bar{\vartheta})\right)^{-1} \frac{\partial F}{\partial \bar{\vartheta}}(x^*(\bar{\vartheta}),\bar{\vartheta})\,.$$

Back-substituting $F = \nabla_x E$ and using the Hessian $H_E(x^*(\bar{\vartheta}),\bar{\vartheta}) := \frac{\partial^2 E}{\partial x^2}(x^*(\bar{\vartheta}),\bar{\vartheta})$ yields

$$\frac{\partial x^*}{\partial \vartheta}(\bar{\vartheta}) = -(H_E(x^*(\bar{\vartheta}),\bar{\vartheta}))^{-1}\frac{\partial^2 E}{\partial\vartheta\partial x}(x^*(\bar{\vartheta}),\bar{\vartheta})\,. \tag{6}$$

The requirement for using (6) from the implicit function theorem is the continuous differentiability of $\partial E/\partial x$ and the invertibility of $H_E$. Applying the chain rule for differentiation the total derivative of the loss function $\mathcal{L}$ of (4) w.r.t. $\vartheta$ is

$$\frac{d\mathcal{L}}{d\vartheta} = -\left[\frac{\partial\mathcal{L}}{\partial x}H_E^{-1}\right]\frac{\partial^2 E}{\partial\vartheta\partial x} + \frac{\partial \mathcal{L}}{\partial\vartheta}\,, \tag{7}$$

where the function evaluation at $(x^*(\bar{\vartheta}),\bar{\vartheta})$ is dropped for brevity. A clever way of setting parentheses, as it is indicated by the squared brackets, avoids explicit inversion of the Hessian matrix. For large problems iterative solvers are required, however.

### 5.2 Derivative of iterative algorithms

We can replace the minimization problem in the lower level of (4) by an algorithm that solves this problem, i.e., the lower level problem is replaced by an equality constraint. This approach shows three advantages: (i) After approximating the lower level of (4) by an algorithm, the approach is exact, in the sense described soon; (ii) the update step of the algorithm can be smooth without the lower level problem to be smooth; (iii) the output is always unique (when an initialization is fixed), i.e., it circumvents the (critical) issue of a non-unique lower level solution.

*Remark 2* The limiting-behavior of the derivative of a sequence of functions—such as the approximation of the lower level solution mapping with an algorithm with increasing iteration number—can differ from that of the sequence of functions itself. Since a rigorous analysis of this (mainly theoretical) fact is hard and we are not going to pursue it in this paper, the proposed approach can be considered heuristic.

*Example 2* The sequence $x^{(n)}(\vartheta) := \vartheta \exp(-n\vartheta^2)$ of functions converges uniformly to the function $x^*(\vartheta) \equiv 0$, $(x^{(n)})'(\vartheta) \to 0$ if $\vartheta \neq 0$ and $(x^{(n)})'(0) \to 1$ as $n \to \infty$. However, obviously the pointwise limit of the derivatives does not coincide with the derivative of the limit function.

Let $\mathcal{A}$ and $\mathcal{A}^{(n)}\colon X \times \mathbb{R}^P \to X$ describe one or $n$ iterations, respectively, of algorithm $\mathcal{A}$ for minimizing $E$ in (4). For simplicity, we assume that the feasible set mapping $\vartheta \mapsto \{x \in \mathbb{R}^N \,|\, (x,\vartheta) \in \operatorname{dom} \mathcal{L}\}$ is constant[4], i.e., the same $X$ is assigned to all $\vartheta \in \mathbb{R}^P$. Note that $X = \mathbb{R}^N$ is permitted.

For a fixed $n \in \mathbb{N}$, we replace (4) by

$$\begin{aligned} &\min_{\vartheta} \ \mathcal{L}(x^*(\vartheta), \vartheta) \\ &s.t. \ x^*(\vartheta) = \mathcal{A}^{(n+1)}(x^{(0)}, \vartheta)\,, \end{aligned} \tag{8}$$

where $x^{(0)}$ is some initialization of the algorithm. The solution map (of the lower level problem) $x^*(\vartheta)$ is the output of the algorithm $\mathcal{A}$ after $n+1$ iterations. If we write down one iteration of the algorithm, i.e., $x^{(n+1)}(\vartheta) = \mathcal{A}(x^{(n)}(\vartheta), \vartheta)$, we have to assume that $x^{(n)}$ depends on the choice of $\vartheta$. However, this dependency can be dropped for the first iterate, which emerges from the initialization.

A suitable algorithm has the properties that $x^{(n)}(\vartheta)$ converges pointwise (for each $\vartheta$) to a solution of the lower level problem as $n$ goes to infinity, i.e., $E(x^{(n)}, \vartheta) = E(\mathcal{A}^{(n)}(x^{(0)}, \vartheta), \vartheta) \to \min_x E(x, \vartheta)$ for $n \to \infty$. As we want to consider also Bregman proximity functions in the algorithm $\mathcal{A}$, the solution for $n \to \infty$ could lie on $\operatorname{bdry}(X)$, despite $x^{(n)} \in \operatorname{int}(X)$ for all $n$. However, this matters only for an asymptotic analysis.

[4] More generally, the concept of outer semi-continuity of the feasible set mapping is needed, otherwise a gradient based method could converge to a non-feasible point.

An interesting aspect of this approach is that, if $\mathcal{A}$ is (totally) differentiable with respect to $\vartheta$, then, by the standard chain rule, $\mathcal{A}^{(n)}$ is differentiable with respect to $\vartheta$ as well. This way, we obtain a totally differentiable approximation to the lower level problem of (4), where the approximation quality can simply be controlled by the number of iterations. For so-called descent algorithms, it holds that

$$E(x^{(n+1)}, \vartheta) - \min_x E(x, \vartheta) \leq E(x^{(n)}, \vartheta) - \min_x E(x, \vartheta)\,.$$

A high number of iterations usually better approximates $\min_x E(x, \vartheta)$ than a small number of iterations. We use "usually" as not all algorithms are descent algorithms, however a sufficiently high number of iterations will always provide a better approximation to $\min_x E(x, \vartheta)$.

Nevertheless, also a small number of iterations is interesting for our approach. Once a certain number of iterations is fixed, the bilevel optimization problem seeks for an optimal performance with exactly this chosen number of iterations. Solving the bilevel optimization problem accurately with a small number of iterations $n$ of the (lower level) algorithm can result in a better performance than a poorly solved (possibly due to local optimal solutions) bilevel problem with a high number of iterations in the lower level.

Our approach is well suited for minimizing the bilevel problem using gradient based methods. The differentiation of $\mathcal{L}$ with respect to $\vartheta$ in (8) is exact; once selected an algorithm no additional approximation is required for computing the derivatives. In contrast, the smoothing approach from Section 5.1 requires the minimization of a smooth objective function, whose solution can only be found approximatively. Therefore, the descent direction that is based on the optimality condition is always erroneous.

The "smoothing parameter" in our approach is the number of iterations of the algorithm that replaces the lower level problem. Since the algorithm's update mapping is assumed to be smooth, i.e., in particular, Lipschitz continuous which formally means

$$\|\mathcal{A}(x, \vartheta) - \mathcal{A}(y, \vartheta)\| \leq \text{const.}\, \|x - y\|\,,$$

the variation of the output after one iterations is limited. Hence a small number of iterations restricts the variation of the output less than a high number of iterations. Additionally, $\mathcal{A}$ is assumed to be differentiable, thus Lipschitz continuous, w.r.t. $\vartheta$. Therefore, intuitively, the higher the number of iterations $n$ the less smoothness of $\mathcal{A}^{(n)}$ can be expected.

Another favorable property is the uniqueness of the algorithm's output.

In terms of leader–follower interpretation of the bilevel optimization problem, we can describe this property as follows. Despite, in general, the leader can not uniquely determine the follower's reaction to his move, he could have an intuition on how the follower is going to compute the moves. If the leader has a good estimation of the follower's strategy, the leader can well plan his own moves.

In order to obtain the derivative of the lower level problem of (8), there are two prominent concepts: forward mode and backward mode. For any vector $\xi \in \mathbb{R}^N$, the *forward mode* corresponds to evaluating the derivative as

$$\xi^\top \frac{dx^{(n+1)}}{d\vartheta}(\vartheta) = \\ \xi^\top \left[ \frac{\partial \mathcal{A}}{\partial x}(x^{(n)}, \vartheta) \frac{dx^{(n)}}{d\vartheta}(\vartheta) \right] + \xi^\top \frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta) \,, \quad (9)$$

whereas the *backward mode/reverse mode* evaluates the derivative as

$$\left( \frac{dx^{(n+1)}}{d\vartheta}(\vartheta) \right)^\top \xi \\ = \left( \frac{dx^{(n)}}{d\vartheta}(\vartheta) \right)^\top \left[ \left( \frac{\partial \mathcal{A}}{\partial x}(x^{(n)}, \vartheta) \right)^\top \xi \right] \\ + \left( \left( \frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)}, \vartheta) \right)^\top \xi \right) \,, \quad (10)$$

where the squared brackets symbolize the different orders of evaluating the terms. In both approaches, replacing and evaluating the term $dx^{(n)}/d\vartheta$ using the preceding iterate $(n-1)$ is done in the respective order.

Mathematically both concepts result in the same solution. However, numerically the approaches are very different. The reverse mode is usually more efficient when the optimization variable $\vartheta$ is high dimensional (i.e., $P$ is large) and the range of the objective function $\mathcal{L}$ is low dimensional—it is always 1 in our setting. This corresponds to $\xi$ being a column vector instead of a derivative matrix. The forward mode is often easier, since it is executed in the same order as the optimization algorithm itself and can be computed online, i.e., during the iteration of the algorithm. However, its downside is that each partial derivative must be initialized and propagated through the iterations. Therefore, the memory requirement is vastly increasing with the dimension $P$. Therefore, we focus on the reverse mode for evaluating the derivatives.

The backward mode is executed in the reverse order of the iterations of the algorithm and needs the optimum $x^*$, which is $x^{(n+1)}$ in our case, for executing the first matrix vector multiplication. All intermediate results toward to optimum must be available. The implementation of the backward mode (10) is shown in Algorithm 1. This approach seems to be quite expensive. However, for a reasonable number of iterations, it is still practical. Nevertheless, in the following we present approximations that reduce the cost significantly.

Usually, evaluating only a few matrix-vector products is sufficient to obtain a high quality approximation of the derivative, i.e., only a few iterations of the reverse mode are required.

In Section 5.3, we motivate another approximation that allows us to evaluate all derivatives at the optimum.

## 5.3 Derivative of fixed point equations

We generalize the result from Section 5.1, where the lower level problem of (4) is replaced by the first-order optimality condition of a smooth approximation. The idea is to consider a different optimality condition. A point is optimal, if it satisfies the fixed point equation of an algorithm $\mathcal{A}\colon X \times \mathbb{R}^P \to X$ solving the original lower level problem, i.e., we address the bilevel problem:

$$\begin{aligned} &\min_{\vartheta} \; \mathcal{L}(x^*(\vartheta), \vartheta) \\ &s.t. \; x^*(\vartheta) = \mathcal{A}(x^*(\vartheta), \vartheta) \,, \end{aligned} \quad (11)$$

where $X \subset \mathbb{R}^N$ is as in Section 5.2 and we have a fixed point $x^*$. This approach is more general than the one in Section 5.1, since we could actually first smoothly approximate the lower level problem and then consider the fixed point equation thereof. On the other hand, for many algorithms and lower level problems both approaches are equivalent, because algorithms are often derived from the first-order optimality condition.

Following the idea of Section 5.2, we can consider a differentiable fixed point equation without the lower level problem to be differentiable. An algorithm that has a differentiable update rule also reveals a differentiable fixed point equation.

Assume that $(x^*, \vartheta)$ solves the fixed point equation. By differentiating the fixed point equation, we obtain

$$\frac{dx}{d\vartheta}(\vartheta) = \frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta), \vartheta) \frac{dx}{d\vartheta}(\vartheta) + \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta), \vartheta) \,,$$

which can be rearranged to yield

$$\frac{dx}{d\vartheta}(\vartheta) = \left( I - \frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta), \vartheta) \right)^{-1} \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta), \vartheta) \,. \quad (12)$$

**Algorithm 1** *Derivative of an abstract algorithm*

```
– Assumptions: $\mathcal{A}$ is totally differentiable.
– Initialization at $n+1$:
    $z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial x}(x^*(\vartheta),\vartheta)\right)^\top \in \mathbb{R}^N$   and   $w^{(n+1)} := 0 \in \mathbb{R}^P$
– Iterations ($n \geq 0$): Update
    for n to 0 :
        $w^{(n)} = w^{(n+1)} + \left(\frac{\partial \mathcal{A}}{\partial \vartheta}(x^{(n)},\vartheta)\right)^\top z^{(n+1)}$
        $z^{(n)} = \left(\frac{\partial \mathcal{A}}{\partial x^{(n)}}(x^{(n)},\vartheta)\right)^\top z^{(n+1)}$
– Final derivative of $\mathcal{L}$ in (8) wrt. $\vartheta$:
    $\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta),\vartheta) = (w^{(0)})^\top + \frac{\partial \mathcal{L}}{\partial \vartheta}(x^*(\vartheta),\vartheta)\,.$
```

Assuming the spectral radius of $(\partial\mathcal{A}/\partial x)(x^*(\vartheta),\vartheta)$ is smaller than 1, we can approximate the inversion using the geometric series:

$$\frac{dx}{d\vartheta}(\vartheta) = \sum_{n=0}^{\infty}\left(\frac{\partial \mathcal{A}}{\partial x}(x^*(\vartheta),\vartheta)\right)^n \frac{\partial \mathcal{A}}{\partial \vartheta}(x^*(\vartheta),\vartheta)\,,$$

where $((\partial\mathcal{A}/\partial x)(x^*(\vartheta),\vartheta))^n$ means the $n$-fold matrix product with itself. Let us approximate this term with a finite summation of $0,\dots,n_0$. Then by a simple rearrangement, for $\xi \in \mathbb{R}^N$, we have (by abbreviating $(\partial\mathcal{A}/\partial x)(x^*(\vartheta),\vartheta)$ by $\partial\mathcal{A}/\partial x$; the same for $\partial\mathcal{A}/\partial\vartheta$):

$$\begin{aligned}\xi^\top \frac{dx}{d\vartheta}(\vartheta) &\approx \xi^\top \sum_{n=0}^{n_0}\left(\frac{\partial \mathcal{A}}{\partial x}\right)^n \frac{\partial \mathcal{A}}{\partial \vartheta}\\ &= \xi^\top \frac{\partial \mathcal{A}}{\partial x}\left(\frac{\partial \mathcal{A}}{\partial \vartheta} + \frac{\partial \mathcal{A}}{\partial x}\left(\frac{\partial \mathcal{A}}{\partial \vartheta} + \dots\right)\right)\\ &= \xi^\top \left[\frac{\partial \mathcal{A}}{\partial x^{(n_0)}}\frac{dx^{(n_0)}}{d\vartheta}\right] + \xi^\top \frac{\partial \mathcal{A}}{\partial \vartheta}\,.\end{aligned}$$

The difference between the last line in this equation and (9) and (10) is the evaluation point of the terms. Where in (9) and (10) the terms for $dx^{(n+1)}/d\vartheta$ are evaluated at $(x^{(n)}(\vartheta),\vartheta)$, here, all terms are evaluated at $(x^*(\vartheta),\vartheta)$. Experimentally, this approximation works also quite well and needs to store only the optimum of the algorithm, which is an immense reduction of the memory requirements.

Summarizing, there are two ways to obtain the derivative of the solution map $x^*$. The first one is by using (12) and to do the matrix inversion, or to solve the arising system of equations when the directional derivative is required as in Section 5.1. The second approach is by approximating the inversion of the matrix with an iterative process like in Section 5.2, where derivatives are here evaluated at the optimum.

### 5.4 Weak differentiation of iterative algorithms

The approach in [11] also considers an algorithm replacing the non-smooth lower level problem. Their underlying methodology, however, is based on weak differentiability, which can be guaranteed for Lipschitz continuous mappings thanks to Rademacher's theorem. If all iteration mappings are Lipschitz continuous with respect to the iteration variable and the parameter $\vartheta$, weak differentiability follows from the chain rule for Lipschitz mappings [17, Theorem 4]. For details, we refer to [11], in particular Section 4. The convergence of the weak derivatives is also left to future work there.

## 6 Explicit derivative for exemplary algorithms

The framework of Bregman proximity functions is key for the idea to approximate a non-smooth optimization problem by an algorithm with smooth update mappings. In this section, we instantiate two such algorithms. Details and examples of Bregman proximity functions are postponed to Section 7.1. For understanding this section, it suffices to know that $D_\psi(x,\bar{x})$ provides a distance measure between two points $x$ and $\bar{x}$, and it can be used to define a Bregman proximity operator $\mathrm{prox}^\psi$ which generalizes the common proximity operator that is based on the Euclidean distance.

### 6.1 Derivative of forward–backward splitting

Let us consider a simple forward–backward splitting [23,30] with Bregman proximity function $D_\psi$ (e.g. [2]). It applies to minimization problems of the form

$$\min_{x\in\mathbb{R}^N} f(x) + g(x)\,,$$

where $f\colon \mathbb{R}^N \to \mathbb{R}$ is a continuously differentiable, convex function with Lipschitz continuous gradient and $g\colon \mathbb{R}^N \to \overline{\mathbb{R}}$ is a proper, lower semi-continuous, convex function with easy-to-evaluate (Bregman) proximity operator. The update rule of the forward–backward splitting that we consider is:

$$\begin{aligned} x^{(n+1)} = \;& \arg\min_{x\in\mathbb{R}^N} g(x;\vartheta) + f(x^{(n)};\vartheta) \\ & + \left\langle \nabla f(x^{(n)};\vartheta), x - x^{(n)} \right\rangle + \frac{1}{\alpha} D_\psi(x, x^{(n)}) \\ =: \;& \mathrm{prox}_{\alpha g}^{\psi}\left(\nabla\psi(x^{(n)}) - \alpha\nabla f(x^{(n)};\vartheta);\vartheta\right) \\ =: \;& \mathrm{prox}_{\alpha g}^{\psi}\left(y^{(n)}(x^{(n)};\vartheta);\vartheta\right), \end{aligned} \tag{13}$$

where we denote $y^{(n)}(x^{(n)};\vartheta) := \nabla\psi(x^{(n)}) - \alpha\nabla f(x^{(n)};\vartheta)$, the intermediate result after the forward step, to simplify the notation in the following. The implementation of the reverse mode for determining the derivative of the solution map of the lower level problem with respect to $\vartheta$ is given in Algorithm 2.

### 6.2 Derivative of primal–dual splitting

Since the primal–dual algorithm with Bregman proximity functions from [8] provides a flexible tool for our purposes, we also want to specify the implementation of the reverse mode for this algorithm. It applies to the convex–concave saddle-point problem

$$\min_x \max_y \langle Kx, y\rangle + f(x) + g(x) - h^*(y)\,,$$

which is derived from $\min_x f(x) + g(x) + h(Kx)$, where $f$ is convex and has a Lipschitz continuous gradient and $g, h$ are proper, lower semi-continuous convex functions with simple proximity operator for $g$ and for the convex conjugate $h^*$.

Let the forward iteration of the primal–dual algorithm with variables $x^{(n)} = (u^{(n)}, p^{(n)}) \in \mathbb{R}^{N_u+N_p}$ be given abstractly as

$$\begin{aligned} u^{(n+1)} &= \mathcal{PD}_u(u^{(n)}, p^{(n)}, \vartheta) \\ &:= \arg\min_u \left\langle \nabla f(u^{(n)}), u - u^{(n)} \right\rangle + g(u) \\ &\qquad + \left\langle Ku, p^{(n)} \right\rangle + \tfrac{1}{\tau} D_u(u, u^{(n)}) \\ p^{(n+1)} &= \mathcal{PD}_p(2u^{(n+1)} - u^{(n)}, p^{(n)}, \vartheta) \\ &:= \arg\min_p h^*(p) - \left\langle K(2u^{(n+1)} - u^{(n)}), p \right\rangle \\ &\qquad + \tfrac{1}{\sigma} D_p(p, p^{(n)})\,, \end{aligned} \tag{14}$$

where $f, g, h, K$ can depend on $\vartheta$. The step size parameter $\tau$ and $\sigma$ must be chosen according to $(\tau^{-1} - L_f)\sigma^{-1} \geq L^2$ where $L = \|K\|$ is the operator norm of $K$ and $L_f$ is the Lipschitz constant of $\nabla f$.

In order to simplify the application of the chain rule throughout the primal–dual algorithm, we show a graphical representation of the information transport in Figure 3, where we use the following abbreviations (analogously for $\mathcal{PD}_p$):

$$\begin{aligned} \mathcal{PD}_u^{(n)} &:= \mathcal{PD}_u(u^{(n)}, p^{(n)}, \vartheta)\,; \\ \mathcal{PD}_p^{(n)} &:= \mathcal{PD}_p(2u^{(n+1)} - u^{(n)}, p^{(n)}, \vartheta)\,; \\ \partial_u \mathcal{PD}_u &:= \frac{\partial \mathcal{PD}_u}{\partial u}\,; \partial_p \mathcal{PD}_u := \frac{\partial \mathcal{PD}_u}{\partial p}\,; \partial_\vartheta \mathcal{PD}_u := \frac{\partial \mathcal{PD}_u}{\partial \vartheta}\,. \end{aligned}$$

*Remark 3* In Section 6.1, we evaluated the forward and the backward step separately using the chain rule. Of course, this could also be done here as well, but would make the understanding more difficult for the reader.

Based on this graphical representation, we can easily write down an implementable algorithm in Algorithm 3.

In order to implement the ergodic primal–dual algorithm whose output is the average of all iterates, i.e., $u^* = \frac{1}{n+1}\sum_{i=0}^n u^{(i)}$, it is useful to concern a running average: denote $s_u^{(n)} := \frac{1}{n+1}\sum_{i=0}^n u^{(i)}$, then $s_u^{(n+1)} = \frac{1}{n+2}u^{(n+1)} + \frac{n+1}{n+2}s_u^{(n)}$. Since taking the derivative is a linear operation, we can simply estimate the derivative for the ergodic primal–dual sequence by averaging all $w^{(n)}$, which can be computed as a running average in the loop of Algorithm 3.

## 7 "Smoothing" using Bregman proximity

Splitting based techniques like those in Section 6 usually handle non-smooth terms in the objective function via a (non-linear/Bregman) proximal step. Sometimes, (convex) conjugation makes terms in the objective amenable for simple and differentiable proximal

```
Algorithm 2  Derivative of a forward–backward splitting algorithm

– Assumptions: $\mathrm{prox}_{\alpha g}^{\psi}$ and $\mathrm{id} + \alpha\nabla f$ are totally differentiable.
– Initialization at $n+1$:

    $z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial x}(x^*(\vartheta), \vartheta)\right)^\top \in \mathbb{R}^N$   and   $w^{(n+1)} := 0 \in \mathbb{R}^P$

– Iterations ($n \geq 0$): Update (where derivatives of $\mathrm{prox}_{\alpha g}^{\psi}$ are evaluated at $(y^{(n)}, \vartheta)$ and
  derivatives of $\nabla f$ at $(x^{(n)}; \vartheta)$)

    for $n$ to $0$:
        $w^{(n)} = w^{(n+1)} + \left(\left(\frac{\partial\,\mathrm{prox}_{\alpha g}^{\psi}}{\partial \vartheta}\right)^\top + \left(-\alpha \frac{\partial(\nabla f)}{\partial \vartheta}\right)^\top \left(\frac{\partial\,\mathrm{prox}_{\alpha g}^{\psi}}{\partial y}\right)^\top\right) z^{(n+1)}$
        $z^{(n)} = \left(\mathrm{id} - \alpha \frac{\partial(\nabla f)}{\partial x}\right)^\top \left(\frac{\partial\,\mathrm{prox}_{\alpha g}^{\psi}}{\partial y}\right)^\top z^{(n+1)}$

– Final derivative of $\mathcal{L}$ in (8) wrt. $\vartheta$:

    $\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta), \vartheta) = (w^{(0)})^\top + \frac{\partial \mathcal{L}}{\partial \vartheta}(x^*(\vartheta), \vartheta)\,.$
```

$$\begin{tikzcd}
 & \frac{du^*}{d\vartheta} \arrow[dl, bend right, "\partial_\vartheta \mathcal{PD}_u^{(n)}"'] \arrow[d, "\partial_u \mathcal{PD}_u^{(n)}"'] \arrow[dr, bend left, "\partial_p \mathcal{PD}_u^{(n)}"] & & \\
d\vartheta & \frac{du^{(n)}}{d\vartheta} \arrow[dl, bend right, "\partial_\vartheta \mathcal{PD}_u^{(n-1)}"'] \arrow[d, "\partial_u \mathcal{PD}_u^{(n-1)}"'] \arrow[dr, "\partial_p \mathcal{PD}_u^{(n-1)}"] & \frac{dp^{(n)}}{d\vartheta} \arrow[l, "2\partial_u \mathcal{PD}_p^{(n-1)}"'] \arrow[dl, "-\partial_u \mathcal{PD}_p^{(n-1)}"'] \arrow[d, "\partial_p \mathcal{PD}_p^{(n-1)}"] \arrow[dr, bend left, "\partial_\vartheta \mathcal{PD}_p^{(n-1)}"] & \\
d\vartheta & \frac{du^{(n-1)}}{d\vartheta} \arrow[dl, bend right, "\partial_\vartheta \mathcal{PD}_u^{(n-2)}"'] \arrow[d, "\partial_u \mathcal{PD}_u^{(n-2)}"'] \arrow[dr, "\partial_p \mathcal{PD}_u^{(n-2)}"] & \frac{dp^{(n-1)}}{d\vartheta} \arrow[l, "2\partial_u \mathcal{PD}_p^{(n-2)}"'] \arrow[dl, "-\partial_u \mathcal{PD}_p^{(n-2)}"'] \arrow[d, "\partial_p \mathcal{PD}_p^{(n-2)}"] \arrow[dr, bend left, "\partial_\vartheta \mathcal{PD}_p^{(n-2)}"] & d\vartheta \\
d\vartheta & \frac{du^{(n-2)}}{d\vartheta} & \frac{dp^{(n-2)}}{d\vartheta} \arrow[l, "2\partial_u \mathcal{PD}_p^{(n-3)}"'] & d\vartheta \\
\vdots & \vdots & \vdots & \vdots
\end{tikzcd}$$

**Fig. 3** This graph shows how the information is backprogated to estimate the derivatives in Algorithm 3. The derivatives at the nodes show what derivative is to be evaluated from this point downwards through the graph. The edges represent multiplicative (transposed) factors. The final derivative is the sum over all leaf nodes.

mappings. Adding the possibility of considering a primal, primal–dual, or dual formulation yields many examples of practical interest.

In the following, we introduce the class of Bregman functions that can be used in combination with the algorithms in Section 6. Then, we discuss a few examples that allow to reformulate several non-smooth terms arising in applications accordingly.

## 7.1 Bregman proximity functions

We consider Bregman proximity functions [4] with the following properties. Let $\psi\colon \mathbb{R}^N \to \overline{\mathbb{R}}$ be a 1-convex function with respect to the Euclidean norm, i.e., it is strongly convex with modulus 1, and denote its domain by $X := \mathrm{dom}\,\psi$. We assume that $\psi$ is continuously differentiable on the interior of its domain $\mathrm{int}(X)$ and continuous on its closure $\mathrm{cl}(X)$.

Then, $\psi$ generates a Bregman proximity function $D_\psi\colon X \times \mathrm{int}(X) \to \mathbb{R}$ by

$$D_\psi(x, \bar{x}) := \psi(x) - \psi(\bar{x}) - \langle \nabla\psi(\bar{x}), x - \bar{x} \rangle\ . \tag{15}$$

**Algorithm 3** *Derivative of a primal–dual algorithm*

```
– Assumptions: $\mathcal{PD}_u$ and $\mathcal{PD}_p$ are totally differentiable.
– Initialization at $n+1$:
    $z^{(n+1)} := \left(\frac{\partial \mathcal{L}}{\partial u}(u^*(\vartheta),\vartheta)\right)^\top \in \mathbb{R}^{N_u}, \quad q^{(n+1)} := 0 \in \mathbb{R}^{N_p}$
    and $w^{(n+1)} := 0 \in \mathbb{R}^P$
– Iterations ($n \geq 0$): Update
    for n to 0:
        $w^{(n)} = w^{(n+1)} + \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial \vartheta}\right)^\top z^{(n+1)} + \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial \vartheta}\right)^\top q^{(n+1)}$
        $q^{(n)} = \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial p}\right)^\top z^{(n+1)} + \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial p}\right)^\top q^{(n+1)}$
        $z^{(n)} = \left(\frac{\partial \mathcal{PD}_u^{(n)}}{\partial u}\right)^\top z^{(n+1)} + 2\left(\frac{\partial \mathcal{PD}_p^{(n-1)}}{\partial u}\right)^\top q^{(n)} - \left(\frac{\partial \mathcal{PD}_p^{(n)}}{\partial u}\right)^\top q^{(n+1)}$
– Final derivative of $\mathcal{L}$ in (8) with $\mathcal{A} = (\mathcal{PD}_u, \mathcal{PD}_p)$ wrt. $\vartheta$:
    $\frac{d\mathcal{L}}{d\vartheta}(u^*(\vartheta),\vartheta) = (w^{(0)})^\top + \frac{\partial \mathcal{L}}{\partial \vartheta}(u^*(\vartheta),\vartheta)\,.$
```

For a sequence $(x^n)_{n\in\mathbb{N}}$ converging to $x \in X$, we require that $\lim_{n\to\infty} D_\psi(x, x^n) = 0$. The 1-convexity of $\psi$ implies that the Bregman function satisfies the inequality

$$D_\psi(x,\bar{x}) \geq \frac{1}{2}\|x-\bar{x}\|^2\,, \quad \forall x \in X\,.$$

These are actually the kind of Bregman proximity functions that are considered in [8]. Obviously $\psi(x) = \frac{1}{2}\|x\|^2$ corresponds to $D_\psi(x,\bar{x}) = \frac{1}{2}\|x-\bar{x}\|^2$.

In iterative algorithms, the Bregman proximity function is used via the (nonlinear/Bregman) proximity operator for a proper, lower semi-continuous, convex function $g\colon X \to \overline{\mathbb{R}}$

$$\operatorname{prox}_{\alpha g}^{\psi}(\bar{x}) := \arg\min_{x\in X}\, \alpha g(x) + D_\psi(x,\bar{x})\,, \tag{16}$$

where we define $\operatorname{prox}_{\alpha g} := \operatorname{prox}_{\alpha g}^{\frac{1}{2}\|\cdot\|^2}$.

There are two kinds of Bregman proximity functions: (i) The function $\nabla\psi$ can be continuously extended to $X$, i.e., $D_\psi$ can be defined on $X \times X$, and (ii) $\psi$ is differentiable on $\operatorname{int}(X)$ (i.e. $\nabla\psi$ cannot necessarily be extended to $\operatorname{cl}(X)$) in which case $D_\psi(x,\bar{x})$ makes sense only on $X \times \operatorname{int}(X)$ and we need to assure that $\operatorname{prox}_{\alpha g}^{\psi}(\bar{x}) \in \operatorname{int}(X)$ for any $\bar{x} \in \operatorname{int}(X)$. For this, we need to assume that $\|\nabla\psi(x)\| \to \infty$ whenever $x$ approaches a boundary point $\operatorname{bdry}(X) := \operatorname{cl}(X) \setminus \operatorname{int}(X)$ (which is sometimes referred to as $\psi$ being essentially smooth [35]).

While evaluating the proximity operator for the first class leads to non-interior methods, the second class generates points that lie in $\operatorname{int}(X)$. As (ii) is an interior method, boundary points can only be attained in the limit. Moreover, for $\bar{x} \in \operatorname{bdry}(X)$, (15) would imply that unless $x = \bar{x}$ the Bregman distance is $+\infty$ for any $x$, which can be represented by $\delta_{[x=\bar{x}]}(x)$. This would mean that $\bar{x} \in \operatorname{bdry}(X)$ is always a fixed point of this Bregman proximity operator, which disables the fixed-point approach in Section 5.3 in this case.

## 7.2 Examples of Bregman functions

Let us consider a few examples, since the concept of Bregman proximity functions is central to the contribution of this paper.

*Example 3* The Euclidean length $\psi(x) = \frac{1}{2}\|x\|_2^2$ is continuously differentiable on the whole space $\mathbb{R}^N$, and therefore, belongs to class (i) of Bregman proximity functions.

*Example 4* The Bregman proximity function generated by $\psi(x) = \frac{1}{2}((x+1)\log(x+1) + (1-x)\log(1-x))$ is defined on the interval $(-1,1)$ and can be continuously extended to $[-1,1]$, and is continuously differentiable on $(-1,1)$ with $|\psi'(x)| \to \infty$ when $x \to \pm 1$. It is 1-strongly convex.

*Example 5* The entropy function $\psi(x) = x\log(x)$, which can be continuously extended to $[x \geq 0]$, is continuously differentiable on $[x > 0]$ with derivative $\psi'(x) =$

$\log(x) + 1$. The derivative cannot be continuously extended to $x = 0$. For $x \to 0$ we have $|\psi'(x)| \to +\infty$. Unfortunately, this function is not even 1-strongly convex on $[x \geq 0]$. However, the function $ax\log(x)$ is 1-strongly convex when restricted to a bounded subset $[0, 1/a]$, $a > 0$. For $a = 1$, the Bregman function $D_\psi(x, \bar{x}) = x(\log(x) - \log(\bar{x})) - (x - \bar{x})$ is generated.

*Example 6* The entropy function can also be used in higher dimensions. Unfortunately, it is hard to assert a simple evaluation of an associated proximity mapping in this case. Consider a polyhedral set $\emptyset \neq X \in \mathbb{R}^N$ given by

$$\begin{aligned} X &= \{x \in \mathbb{R}^N \,|\, \forall i = 1, \ldots, M \colon \langle a_i, x\rangle \leq b_i\} \\ &= \bigcap_{i=1}^{M} \{x \in \mathbb{R}^N \,|\, \langle a_i, x\rangle \leq b_i\} \end{aligned}$$

for vectors $0 \neq a_i \in \mathbb{R}^N$, and $b_i \in \mathbb{R}^M$, $i = 1, \ldots, M$. Then, the generating function

$$\psi(x) = \sum_{i=1}^{M} (b_i - \langle a_i, x\rangle) \log(b_i - \langle a_i, x\rangle)$$

is designed such that for any point $\bar{x} \in \mathrm{int}(X)$ any other point $x \notin X$ is “moved infinitly far away” with respect to the Bregman distance $D_\psi(x, \bar{x})$. Therefore $\|\nabla\psi(x)\| \to \infty$ for $x$ tending towards a point on the boundary $\mathrm{bdry}(X)$. Nevertheless, $\psi$ is continuous on $X$ and strongly convex, if $X$ is bounded.

## 7.3 Examples of smooth Bregman proximity operators

The Bregman proximity functions that we presented are particularly interesting if the evaluation of the proximal mapping (16) is a constrained minimization problem, i.e. the involved function $g$ in $\mathrm{prox}_g^\psi$ is extended-valued and $+\infty$ outside the constraint (closed) convex set $X \subset \mathbb{R}^N$. The Bregman function can replace or simplify the constraint set. In the following, we consider a few examples of practical interest. Thanks to (convex) conjugation the class of functions that are amenable to our approach is big.

We consider a basic class of functions $g(x) = \langle x, c\rangle + \delta_X(x)$ for some $c \in \mathbb{R}^N$. The associated (non-linear) proximity operator from (16) is given by

$$\mathrm{prox}_{\alpha g}^\psi(\bar{x}) = \arg\min_{x \in X} \alpha\, \langle x, c\rangle + D_\psi(x, \bar{x}) \,.$$

The corresponding (necessary and sufficient) optimality condition, which has a unique solution, is

$$\begin{aligned} &0 \in c + \nabla\psi(x) - \nabla\psi(\bar{x}) + \partial\delta_X(x) \\ \Leftrightarrow\ &\nabla\psi(\bar{x}) - c \in \nabla\psi(x) + \mathrm{N}_X(x) \,, \end{aligned}$$

where $\mathrm{N}_X(x)$ denotes the normal cone at $x$ of the set $X$. Suppose $\bar{x} \in \mathrm{int}(X)$. If $\psi$ is chosen such that $\|\nabla\psi(x)\| \to +\infty$ for $x \to \tilde{x} \in \mathrm{bdry}(X)$, then the solution of the proximal mapping is in $\mathrm{int}(X)$. Since $\mathrm{N}_X(x) = 0$ for $x \in \mathrm{int}(X)$, the optimality condition simplifies to

$$\nabla\psi(\bar{x}) - c = \nabla\psi(x) \,, \tag{17}$$

i.e. the constraint is taken care of by the Bregman proximity function. Summarizing, the goal of our approach (for this basic function $g$) consists of determining $\psi$, respectively $D_\psi$, such that

- the constraint set can be neglected,
- (17) can be solved efficiently (possibly in closed form),
- and, due to the gradient computation of the algorithm mapping (and the chain rule), the solution function of (17) is required to be differentiable wrt. $x$ and $\vartheta$, where possibly $c = c(\vartheta)$.

*Example 7* For a linear function $g(x) = \langle c, x\rangle + \delta_{[x \geq 0]}(x)$ the entropy function from Example 5 can be summed-up for each coordinate to get rid of the non-negativity constraint. The proximity operator reads:

$$\left(\mathrm{prox}_{\alpha g}^{\sum_j x_j \log x_j}(\bar{x})\right)_i = \bar{x}_i \exp(-\alpha c_i) \,.$$

A closer look at the iterations of the forward–backward splitting (FBS) algorithm (13) reveals that this situation arises for example with $c = \nabla f(\bar{x})$, i.e. in the iterations of FBS for the minimization of

$$\min_{x \in \mathbb{R}^N} f(x) + \delta_{[x \geq 0]}(x) \,.$$

A particular instance of this problem is the *non-negative least squares problem*, i.e. $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ with a matrix $A$ and a vector $b$.

*Example 8* The most frequent application of the entropy-prox is for the minimization of a linear function $g(x) = \langle c, x\rangle$ over the unit simplex in $\mathbb{R}^N$. Since the entropy moves negative values infinitly far away, projecting a point $\bar{x} \in \mathbb{R}_+^N$ onto the unit simplex $\{x \in \mathbb{R}^N \,|\, \sum_{i=1}^N x_i = 1 \text{ and } x_i \geq 0\}$ reduces to the projection onto the affine subspace $\{x \in \mathbb{R}^N \,|\, \sum_{i=1}^N x_i = 1\}$, which can be given in closed-form, i.e.,

$$\left(\mathrm{prox}_{\alpha g}^{\sum_j x_j \log x_j}(\bar{x})\right)_i = \frac{\bar{x}_i \exp(-\alpha c_i)}{\sum_{j=1}^N \bar{x}_j \exp(-\alpha c_j)} \,.$$

This proximal problem arises for example in the *multi-label segmentation problem* in Section 9.1 or in *Matrix games* (see [8, Section 7.1]).

*Example 9* For the function $g(x) = \langle c, x\rangle + \delta_{[-1\leq x\leq 1]}(x)$ the Bregman function from Example 4 reduces the minimization problem in the proximal mapping to an unconstrained problem. The proximal mapping with $\psi(x) = \sum_i \frac{1}{2}((x_i+1)\log(x_i+1) + (1-x_i)\log(1-x_i))$ reads:

$$\left(\mathrm{prox}^{\psi}_{\alpha g}(\bar x)\right)_i = \frac{\exp(-2\alpha c_i) - \frac{1-\bar x_i}{1+\bar x_i}}{\exp(-2\alpha c_i) + \frac{1-\bar x_i}{1+\bar x_i}}\,.$$

Obviously, this example can be easily adjusted to any Cartesian product of interval constraints. The importance of this exemplary function $g$ becomes clear in the following.

Functions that are linear on a constraint set also arise when conjugate functions are considered. For instance the $\ell_1$-norm can be represented as

$$\|x\|_1 = \max_y \,\langle x, y\rangle + \delta_{[-1\leq y\leq 1]}(y)\,,$$

which in combination with the primal–dual (PD) algorithm (14) results in subproblems (cf. the update step of the dual variable in (14)) of the type discussed in the preceding examples, here Example 9. From this perspective, optimization problems involving a linear operator $\mathcal{D}$ and the $\ell_1$-norm $\|\mathcal{D}x\|_1$ are as easy to address.

This idea of conjugation can be put into a slightly larger framework, as the convex conjugate of any positively one-homogeneous proper, lsc, convex function is an indicator function of a closed convex set. Unfortunately, we must assume that projecting onto such a set is easy ("prox-friendliness"). Therefore, the following example is restricted to the (additively) separable case.

*Example 10* Let $g$ be a (additively) separable, positively one-homogeneous, proper, lsc, convex functions $g(x) = \sum_{i=1}^N g_i(x_i)$. Thanks to its properties $g$ coincides with its bi-conjugate function $g^{**}$ and we can consider

$$g(x) = g^{**}(x) = \sum_{i=1}^N \max_{y_i}\, x_i y_i - \delta_{Y_i}(y_i)\,,$$

where $Y_i = [a_i, b_i]$ is a closed interval in $\mathbb{R}$. Again the dual update step of (14) involves problems of type of Example 9 with $h^*(y) = \sum_i \delta_{Y_i}(y_i)$.

### 7.4 Example applications

Finally, we present a few examples from image processing, computer vision and machine learning that allow for an algorithm with a smooth update mapping though the minimization problem is non-smooth. These problems might occur as lower level of a bilevel optimization problem.

*Example 11* A potentially interesting minimization problem for the task of denoising of a vector $b$ (which could represent an image) is the following:

$$\min_{x\in\mathbb{R}^N}\ \frac{1}{2}\|x-b\|_2^2 + \lambda\sum_{i=1}^{N_\alpha}\alpha_i\,\rho\left(\sum_{j=1}^{N_\beta}\beta_{i,j}B_{i,j}x\right)\,,$$

with $\vartheta = (\lambda, \alpha_1, \dots, \alpha_{N_\alpha}, \beta_{1,1}, \dots, \beta_{N_\alpha,N_\beta})^\top$, a collection of basis filters (linear mappings) $B_{i,j}$, and a convex penalty function $\rho\colon \mathbb{R}\to\mathbb{R}_+$. The problem where $\rho(x) = |x|$ is considered in [22,9], however only a smooth approximation is solved in the end. Using our idea, we can apply the primal–dual algorithm (14) like in Example 10.

This parameter learning problem can easily be extended to more complicated "data-terms" as required for instance in deblurring, i.e., $\frac{1}{2}\|x-b\|_2^2$ can be replaced by $\frac{1}{2}\|Ax-b\|_2^2$ with a matrix $A$ that implements a convolution of a filter $k_A$ with $x$.

Moreover, the data-term could also be replaced by a robust (non-smooth) $\ell_1$-norm. Using the primal–dual algorithm (14) we can consider again the dualized version and thereby reduce the evaluation of the proximity operator to a constrained problem with a simple set like in Example 10.

If, even more flexibility is desired, also the data term can be parametrized, e.g., for learning the noise model. Learning the noise model with a (simple) total variation penalty is pursued in [34] by smoothing.

## 8 Toy example

The bilevel problem that we consider here is a parameter learning problem of a one dimensional non-negative least-squares problem:

$$\begin{aligned}&\min_{\vartheta\in\mathbb{R}}\ \frac{1}{2}(x^*(\vartheta) - \mathfrak{g})^2\\ &\qquad s.t.\ x^*(\vartheta) = \arg\min_{x\in\mathbb{R}}\ \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 + \delta_{[x\geq 0]}(x)\,,\end{aligned} \tag{18}$$

where $\vartheta$ is the optimization variable of the bilevel problem, $b\in\mathbb{R}$ is the input of the least squares problem, and $\lambda$ is a positive weighting parameter. Given $\vartheta$ and $b$ the lower level problem solves the non-negative least squares problem. The squared Euclidean loss function in the upper level problem compares the output of the lower level problem for some $\vartheta$ and $b$ to the ground truth $\mathfrak{g} := x^*(\vartheta^*)$, which is generated by solving the lower level problem with some predefined value $\vartheta^*$. The goal of the bilevel optimization problem is to find $\vartheta^*$ given $b$ and $\mathfrak{g}$.

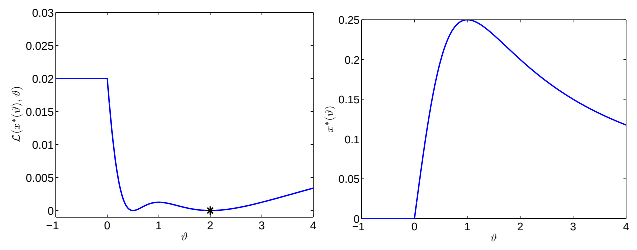


**Fig. 4** Visualization of the loss function $\mathcal{L}(x(\vartheta),\vartheta)$ for the 1D example (18) on the left side. The optimum is marked with a black star. On the right hand side, the solution map of the lower level problem is shown.

The analytic solution of the lower level problem (the solution map) is

$$x^*(\vartheta) = \max\left(0, \frac{\lambda\vartheta b}{1+\lambda\vartheta^2}\right)$$

and is shown on the right hand side of Figure 4. It is obviously a non-smooth function with a non-differentiable point at $\vartheta = 0$. Plugging the solution map into the upper level problem shows the actual objective to be minimized; see the left hand side of Figure 4.

## 8.1 Experimental setup

In the following experiments, we numerically explore the gradients computed with the proposed techniques. We do not consider the actual minimization of the bilevel problem. The computed gradients could be used by any (non-convex) first-order gradient based method.

*Analytic subdifferential.* For $\vartheta \neq 0$ the standard chain rule from calculus can be applied and we can directly write down the derivative of the whole problem, namely

$$\frac{d\mathcal{L}}{d\vartheta}(x(\vartheta)) = \frac{\lambda b(1-\lambda\vartheta^2)}{(1+\lambda\vartheta^2)^2}(x(\vartheta)-\mathfrak{g})\,.$$

For $\vartheta = 0$, we consider the derivative

$$\frac{d\mathcal{L}}{d\vartheta}(x(\vartheta)) = [0, \lambda b(x(0)-\mathfrak{g})]\,,$$

where $[0,\lambda b]$ is replaced by $[\lambda b, 0]$ if $\lambda b < 0$.

*Implicit differentiation approach Section 5.1.* In order to apply this technique, we first need to smooth the lower level problem. Since we want to avoid solutions $x^*(\vartheta) = 0$, we introduce a log-barrier and replace the lower level problem by

$$f_\mu(x,\vartheta) := \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 - \mu\log(x)$$

for some small $\mu > 0$. Thus, we can drop the non-negativity constraint. In order to compute the gradient via the implicit differentiation formula (7), we need to minimize $f_\mu$ with respect to $x$ and compute the second derivatives (we abbreviate the $x$-derivative with $f'_\mu$ and $\vartheta$-derivative with $\partial_\vartheta f_\mu$)

$$\begin{aligned} f'_\mu(x,\vartheta) &= \lambda\vartheta(\vartheta x - b) + x - \frac{\mu}{x}\,; \\ f''_\mu(x,\vartheta) &= \lambda\vartheta^2 + 1 + \frac{\mu}{x^2}\,; \\ \partial_\vartheta f'_\mu(x,\vartheta) &= 2\lambda\vartheta x - \lambda b\,. \end{aligned} \tag{19}$$

Then, (7) yields

$$\frac{d\mathcal{L}}{d\vartheta}(x^*(\vartheta)) = -(x(\vartheta) - \mathfrak{g})(f''_\mu(x^*(\vartheta),\vartheta))^{-1}\partial_\vartheta f'_\mu(x^*(\vartheta),\vartheta)\,.$$

This approach is denoted `Smoothed-impl` in the following.

*Algorithmic differentiation approach Section 5.2.* We consider two algorithms: projected gradient descent and forward–backward splitting with Bregman proximity functions. Both algorithms are splitting methods that distribute the objective into a smooth function $f$ and a nonsmooth function $g$, which for our example reads

$$f(x,\vartheta) = \frac{\lambda}{2}(\vartheta x - b)^2 + \frac{1}{2}x^2 \quad \text{and} \quad g(x) = \delta_{[x\geq 0]}(x)\,.$$

Projected gradient descent operates by a gradient descent step with respect to the smooth function $f$ followed by a projection onto the (convex) set $[x \geq 0]$:

$$\begin{aligned} x^{(n+1)} &= \operatorname{proj}_{[x\geq 0]}(x^{(n)} - \alpha f'(x^{(n)},\vartheta)) \\ &= \max(0, x^{(n)} - \alpha f'(x^{(n)}))\,. \end{aligned} \tag{20}$$

Note that the projection onto the convex set can also be interpreted as solving the proximity operator associated with the function $g$.

The second algorithm is obtained by changing the distance function for evaluating the proximity operator to the Bregman distance from Example 5 (in the appendix). It results in

$$x^{(n+1)} = x^n \exp(-\alpha f'(x^{(n)},\vartheta))\,. \tag{21}$$

As we assume that $x^0 \in [x > 0]$ the Bregman proximity function takes care to not leave the feasible set. The back-projection can be dropped.

In order to apply Algorithm 1 or 2, we need to compute the second derivatives of the update steps (20) and (21). The second derivatives of $f = f_\mu$ with $\mu = 0$ are given in (19). Although, actually (20) is not differentiable, it is differentiable almost everywhere, and in the experiment, we formally applied the chain rule

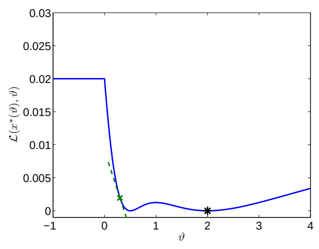


**Fig. 5** Analytic tangents to the upper level objective function of (18) at $\vartheta = 0.3$ and $\vartheta = 0$. The function is non-smooth and, thus, at $\vartheta = 0$ there exists many tangent lines.

and assigned an arbitrary subgradient wherever it is not unique, i.e.,

$$\frac{\partial\,\mathrm{proj}_{[x\geq 0]}}{\partial x}(x,\vartheta) = \begin{cases} 0, & \text{if } x<0\,;\\ 1, & \text{if } x>0\,;\\ [0,1], & \text{if } x=0\,; \end{cases}$$

and $\frac{\partial\,\mathrm{proj}_{[x\geq 0]}}{\partial\vartheta} = 0$. In the following experiments, this approach is denoted `Proj.GD`.

For (21), we use Algorithm 1 and obtain[5]

$$\begin{aligned} \frac{\partial\mathcal{A}}{\partial\vartheta}(x^{(n)},\vartheta) =&\; -\alpha x\exp(-\alpha f'(x^{(n)},\vartheta))\frac{\partial f'}{\partial\vartheta}(x^{(n)},\vartheta)\\ \frac{\partial\mathcal{A}}{\partial x}(x^{(n)}.\vartheta) =&\; \exp(-\alpha f'(x^{(n)},\vartheta))\\ &\; -\alpha x^{(n)}\exp(-\alpha f'(x^{(n)},\vartheta))f''(x^{(n)},\vartheta)\,. \end{aligned}$$

This approach is denoted `Bregman-FB`.

*Implicit differentiation of fixed points approach Section 5.3.* As explained above, the idea to directly differentiate the fixed point equation of an algorithm, implies two techniques. One is by applying Algorithm 1 to (21) but evaluating all derivatives at the optimum (denoted `Bregman-FB2`), and the other is to do the numerical inversion as in (12) (denoted `Bregman-FB-impl`).

## 8.2 Analysis of the 1D example

In the experiments, we focus on the estimation of the gradient (in Figure 5). Therefore, the step size parameters of the individual algorithms are chosen such that a comparable convergence of the lower level energy is achieved, if possible.

For `Proj.GD`, `Bregman-FB`, and `Bregman-FB2` the chain rule must be applied recursively. We plot the change of the gradient accumulation along these back-iterations (of 200 forward-iterations) in bottom of Figure 6 and the energy evolution in the upper part of this figure. In this example, we can observe a linear

[5] Note that we kept the order of the terms given by the chain rule, since for multi-dimensional problems the products are matrix products and are, in general, not commutative.

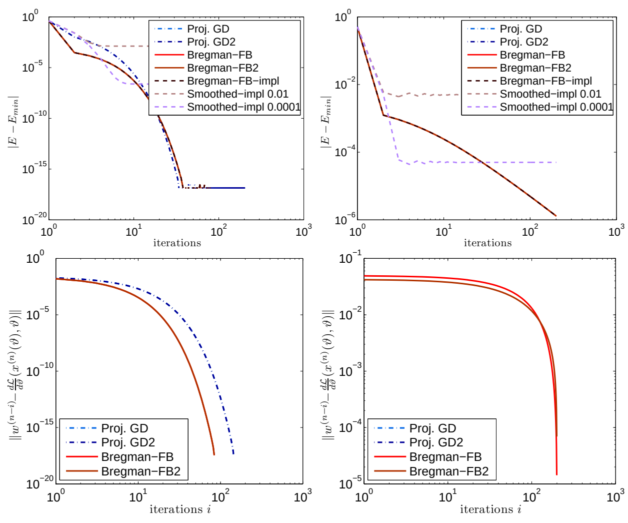


**Fig. 6** The upper row shows the energy decrease along the forward iterations, and the lower row the convergence to the respective gradient value along the back-iterations. On the left hand side the plot is generated with $\vartheta = 0.3$ and on the right hand side with $\vartheta = 0$. The "-impl" methods do not appear in the bottom row as no back-iterations are involved. For $\vartheta = 0$, due to the simple structure of the lower level problem, projected gradient descent converges exactly in one iteration, thus it is not shown. The gradient converges linearly to its final value, which means that often a few back-iterations are enough to achieve a gradient of good quality.

decrease in the contribution to the respective final gradient value, which shows that back-iterations can be stopped after a few iterations without making large errors. In low dimensional examples 10–20 back-iterations are appropriate; for higher dimensional ones 50–100 back-iterations are usually good. However, this depends on the type of problem, the convergence of the forward-iterations, and needs to be explored in more depth in the future.

An interesting aspect of this experiment is that the approximations `Bregman-FB2` and `Proj.GD2` seems to be good, as they show the same gradient accumulation as `Bregman-FB` and `Proj.GD`, respectively. This situation changes when the number of forward-iterations are reduced. For about 15 forward-iterations, a difference of order $10^{-4}$ becomes visible (case $\vartheta = 0.3$).

Now, we address the actual convergence of the gradient towards the analytic gradient, with respect to different approximation accuracies (varied by the number of back-iterations). Figure 7 shows the convergence for $\vartheta = 0.3$ and Figure 8 for $\vartheta = 0$. Numerically, we observe convergence to the analytic gradients. As we mentioned before, a theoretical asymptotic analysis is missing. Surprisingly, all methods perform equally well in the case $\vartheta = 0$. The estimated gradient lies always in the subdifferential at this point. The range of the subdifferential is indicated with bright green color in Figure 8. While

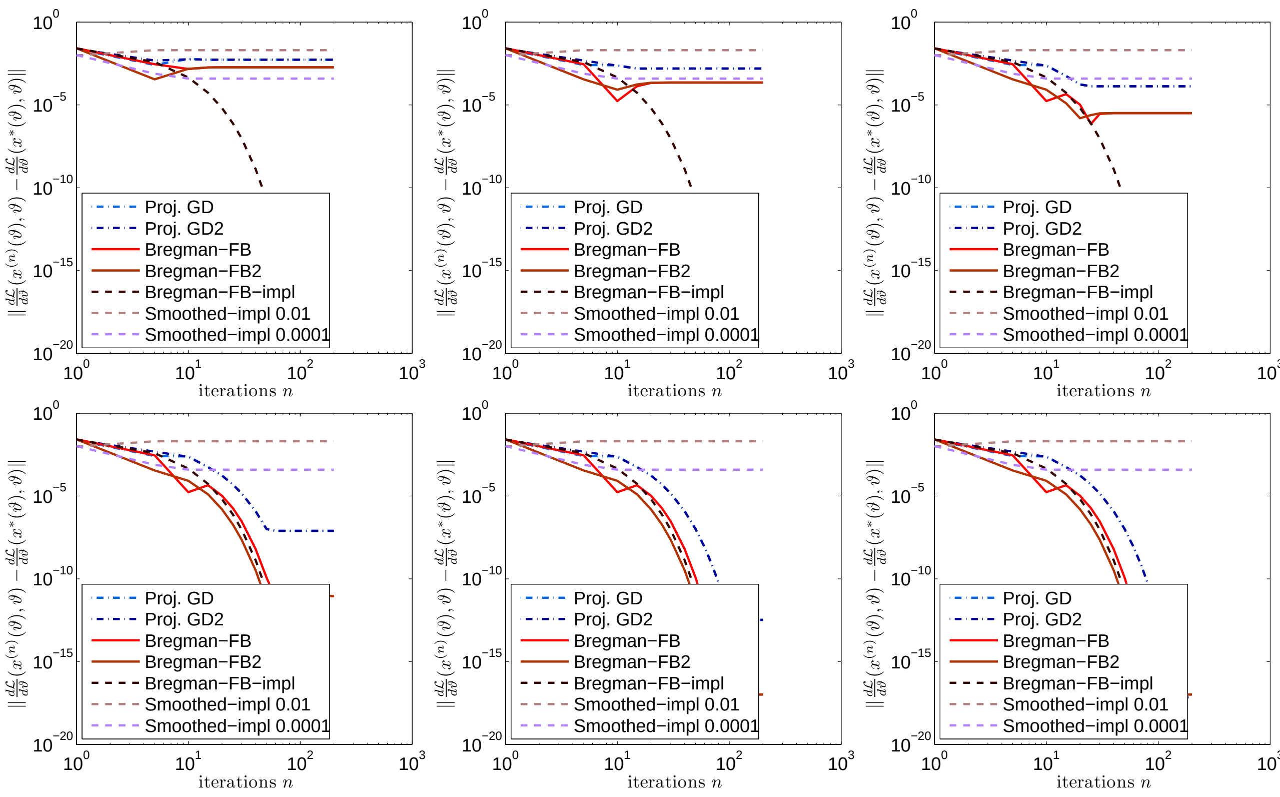


**Fig. 7** Convergence of the numerical gradients towards the analytic gradient for $\vartheta = 0.3$. Row-wise, from left to right, the number of back-iterations is increased: 5, 10, 20, 50, 100, 200. Obviously, the more back-iterations, the more accurate the computed gradient. The "-impl" methods always perform equally, as no back-iterations are required. `Smoothed-impl` performs worst due to the rough approximation. Our methods `Bregman-FB`, `Bregman-FB2`, and `Bregman-FB-impl` are the best; and converge slightly better than `Proj.GD` and `Proj.GD2`.

`Proj.GD` and `Proj.GD2` estimates a gradient from the boundary of the subdifferential, the other methods estimate a subgradient from the interior. However, all of these values are feasible and belong to the analytic subdifferential.

## 9 Application to Multi-Label Segmentation

In this section, we show how the developed abstract idea can be applied in practice. Before the actual bilevel learning problem is presented, we introduce a multi-label segmentation model. We use a convolutional neural network (CNN) to parametrize the segmentation model. Alternatively, this construction can be thought of as having a segmentation model as the final stage of a deep neural network. In this setting, the bi-level problem then amounts to finding the parameters of the CNN such that the loss on training data is minimized. The presented approach provides a generic recipe to train such systems in an end-to-end fashion.

### 9.1 Model

Given a cost tensor $\mathfrak{c} \in X^{N_l}$, where $X = \mathbb{R}^{N_x N_y}$, that assigns to each pixel $(i,j)$ and each label $k$, $i = 1,\dots,N_x$, $j = 1,\dots,N_y$, $k = 1,\dots,N_l$, a cost $\mathfrak{c}^k_{i,j}$ for the pixel taking label $k$. We often identify $\mathbb{R}^{N_x\times N_y}$ with $\mathbb{R}^{N_x N_y}$ by $(i,j) \mapsto i + (j-1)N_x$ to simplify the notation. The sought segmentation $u \in X^{N_l}_{[0,1]}$, where $X_{[0,1]} = [0,1]^{N_x N_y} \subset X$, is represented by a binary vector for each label. As a regularizer for a segment's plausibility we measure the boundary length using the total variation (TV). The discrete derivative operator $\nabla\colon X \to Y$, where we use the shorthand $Y := X \times X$ (elements from $Y$ are considered as column vectors), is defined as (let the pixel dimension be $1\times 1$):

$$
\begin{aligned}
(\nabla u^k)_{i,j} &:= \begin{pmatrix} (\nabla u^k)^x_{i,j} \\ (\nabla u^k)^y_{i,j} \end{pmatrix} \in Y (= \mathbb{R}^{2N_x N_y}), \\
\mathcal{D}u &:= (\nabla u^1, \dots, \nabla u^{N_l}), \\
(\nabla u^k)^x_{i,j} &:= \begin{cases} u^k_{i+1,j} - u^k_{i,j}\,, & \text{if } 1 \le i < N_x, 1 \le j \le N_y \\ 0\,, & \text{if } i = N_x, 1 \le j \le N_y \end{cases}
\end{aligned}
$$

$(\nabla u^k)^y_{i,j}$ is defined analogously. From now on, we work with the image as a vector indexed by $\mathbf{i} = 1,\dots,N_x N_y$.

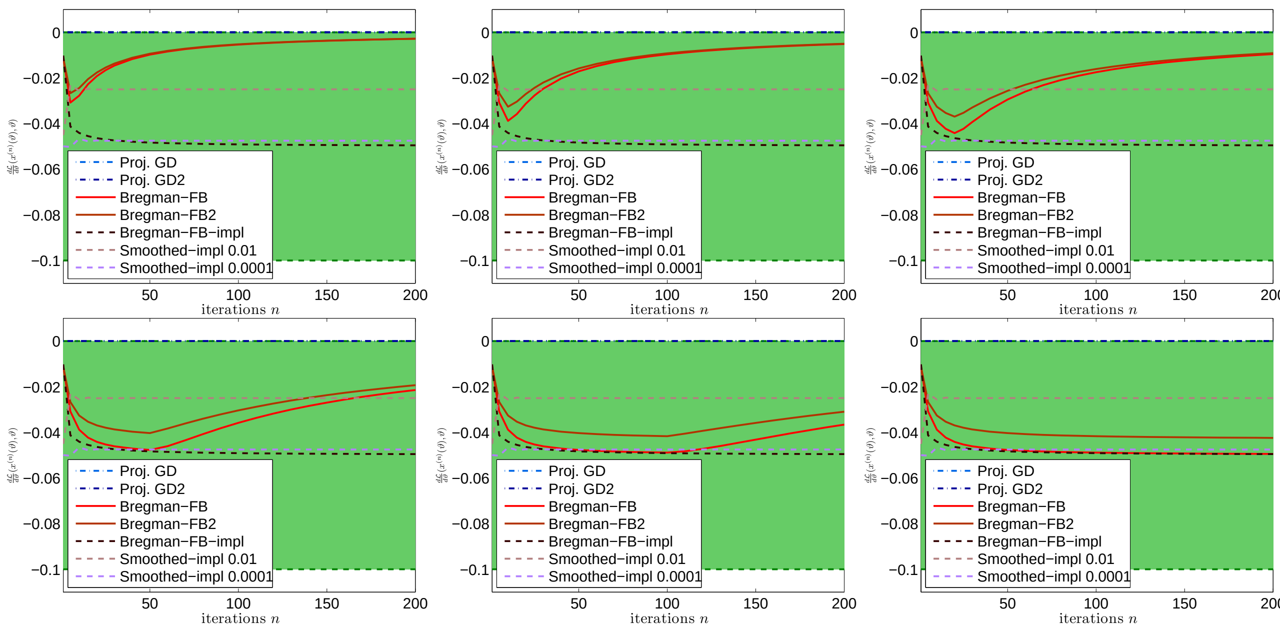


**Fig. 8** Convergence of the numerical gradients towards the analytic gradient for $\vartheta = 0$. Row-wise, from left to right, the number of back-iterations is increased: 5, 10, 20, 50, 100, 200. All methods perform equally well, as they lie in the bright green area that indicates the range of the subdifferential.

Let elements in $Y$ be indexed with $\mathbf{j} = 1, \dots, 2N_xN_y$. Let the inner product in $X$ and $Y$ be given, for $u^k, v^k \in X$ and $p^k, q^k \in Y$, as:

$$\left\langle u^k, v^k \right\rangle_X := \sum_{\mathbf{i}=1}^{N_xN_y} u_{\mathbf{i}}^k v_{\mathbf{i}}^k, \; \left\langle p^k, q^k \right\rangle_Y := \sum_{\mathbf{j}=1}^{2N_xN_y} p_{\mathbf{j}}^k q_{\mathbf{j}}^k,$$

$$\langle u, v\rangle_{X^{N_l}} := \sum_{k=1}^{N_l} \left\langle u^k, v^k \right\rangle_X, \; \langle p, q\rangle_{Y^{N_l}} := \sum_{k=1}^{N_l} \left\langle p^k, q^k \right\rangle_Y.$$

The (discrete, anisotropic) TV norm is given by $\|\mathcal{D}u\|_1 := \sum_{k=1}^{N_l} \sum_{\mathbf{j}=1}^{2N_xN_y} |(\nabla u^k)_{\mathbf{j}}|$, where $|\cdot|$ is the absolute value. In the following, the iteration variables $\mathbf{i} = 1, \dots, N_xN_y$ and $\mathbf{j} = 1, \dots, 2N_xN_y$ always run over these index sets, thus we drop the specification; we adopt the the same convention for $k = 1, \dots, N_l$. We define the pixel-wise nonnegative unit simplex

$$\begin{aligned} \varDelta^{N_l} := \{\forall(\mathbf{i}, k) \colon 0 \le u_{\mathbf{i}}^k \le 1 \\ \text{and } \forall \mathbf{i} \colon \textstyle\sum_k u_{\mathbf{i}}^k = 1 \; u \in X^{N_l}\}\,, \end{aligned} \tag{22}$$

and the pixel-wise (closed) $\ell_\infty$-unit ball around the origin

$$B_1^{\ell_\infty}(0) := \{p \in Y^{N_l} \,|\, \forall(\mathbf{j}, k) \colon |p_{\mathbf{j}}^k| \le 1\}\,.$$

Finally, the segmentation model reads

$$\min_{u \in X^{N_l}} \; \langle \mathfrak{c}, u\rangle_{X^{N_l}} + \|\mathcal{W}\mathcal{D}u\|_1\,, \quad s.t.\; u \in \varDelta^{N_l}\,, \tag{23}$$

where we use a diagonal matrix $\mathcal{W}$ to support contrast-sensitive penalization of the boundary length.

This model and the following reformulation as a saddle-point problem are well known (see e.g. [7])

$$\begin{aligned} \min_{u \in X^{N_l}} \max_{p \in Y^{N_l}} \; \langle \mathcal{W}\mathcal{D}u, p\rangle_{Y^{N_l}} + \langle u, \mathfrak{c}\rangle_{X^{N_l}}\,, \\ s.t.\; u \in \varDelta^{N_l},\; p \in B_1^{\ell_\infty}(0)\,. \end{aligned} \tag{24}$$

The saddle-point problem (24) can be solved using the ergodic primal-dual algorithm [8], which leads to an iterative algorithm with totally differentiable iterations. The primal update in (14) is discussed in Example 8 and the dual update of (14) is essentially Example 9. As a consequence Algorithm 3 can be applied to estimate the derivatives. A detailed derivation of the individual steps of the algorithm can be found in [29].

## 9.2 Parameter Learning

We consider (23) where the cost $\mathfrak{c}$ is given by the output of a CNN which takes as input an image $\mathfrak{I} \in X^{N_c}$ to be segmented and is defined via a set of weights $\vartheta$. Formally, we have $\mathfrak{c}_{\mathbf{i}}^k = \mathfrak{f}_{\mathbf{i}}^k(\vartheta, \mathfrak{I})$ with $\mathfrak{f} : \mathbb{R}^{N_\vartheta} \times X^{N_c} \to X^{N_l}$, where $N_c$ denotes the number of channels of the input image and $N_\vartheta$ is the number of weights parametrizing the CNN.

The training set consists of $N_T$ images $\mathfrak{I}^1, \dots, \mathfrak{I}^{N_T} \in X^{N_c}$ and their corresponding ground truth segmentations $\mathfrak{g}^1, \dots, \mathfrak{g}^{N_T} \in \{1, \dots, N_l\}^{N_xN_y}$.

In order to find the parameters $\vartheta$ of the CNN, we consider an instance of the general bilevel optimization

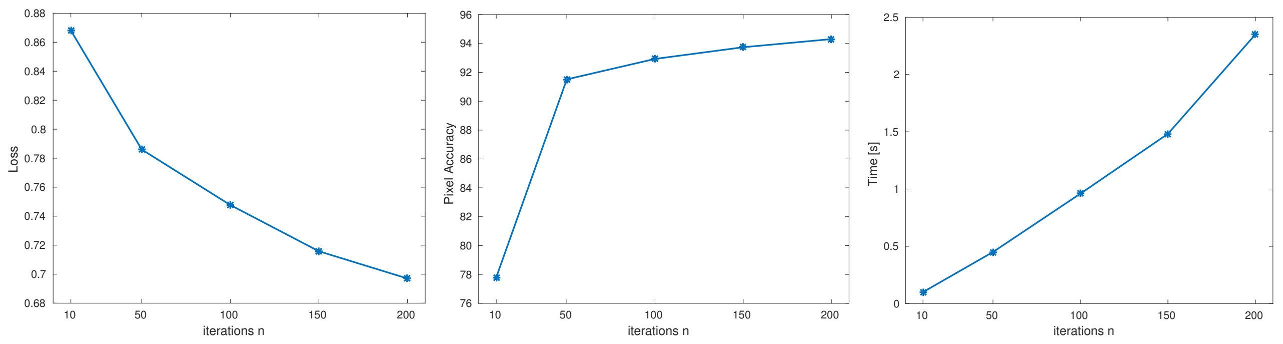


**Fig. 9** Training error vs. number of iterations of the algorithm solving the lower level problem. From left to right, average per-pixel loss, per-pixel accuracy and time per outer iteration. The timing includes the forward pass as well as the gradient computations. Timings were taken on a NVIDIA Geforce Titan X GPU. A higher number of iterations clearly leads to lower error, but comes at the cost of a higher computational complexity.

problem (4):

$$\min_{\vartheta \in \mathbb{R}^{N_\vartheta}} \sum_{t=1}^{N_T} \sum_{\mathbf{i}=1}^{N_x N_y} \log \Big( \sum_{k=1}^{N_l} \exp(u_{\mathbf{i}}^k(\vartheta, \mathfrak{I}^t)) \Big) - \mathfrak{g}_{\mathbf{i}}^t(\vartheta, \mathfrak{I}^t)$$
$$s.t.\ u(\vartheta, \mathfrak{I}^t) = \arg \min_{u \in X^{N_l}} E(u, \mathfrak{f}(\vartheta, \mathfrak{I}^t)), \tag{25}$$

where energy $E$ in the lower level problem is (23) and the higher-level problem is defined as the softmax loss.

*Remark 4* We could equivalently use a multinomial logistic loss, since $u_{\mathbf{i}}(\vartheta, \mathfrak{I}^t)$ lies in the unit simplex by construction. We use this definition in order to allow for a simplified treatment of the case of training a CNN without the global segmentation model.

## 9.3 Experiments

We implemented our approach as a custom layer in the MatConvNet framework [38]. We use the Stanford Background dataset [19], which consists of 715 input images and pixel-accurate ground truth consisting of the geometric classes *sky*, *vertical* and *horizontal* for our experiments and use ADAM [21] for the minimization of the higher-level problem. We found that in general plain stochastic gradient descent performs poorly in our setting, since the global segmentation model can lead to vanishing gradients.

In a first experiment we use a small subset of 9 images from the dataset to show the influence of the number of iterations used to solve the lower-level problem (23) on the training objective. We learn a small network consisting of four layers of alternating convolutions with a kernel width of 3 pixels and ReLU units followed by a fully connected layer. We add $3 \times 3$ max-pooling layers with a stride of two after the first and the second convolutional layers, which effectively downsamples the responses by a factor of 4. We add a deconvolutional layer to upsample the responses to the original image size. The penultimate layer of the CNN consist of a multiplicative scaling (analogous to a scalar smoothness parameter) of the CNN output followed by the global segmentation model (23). We run ADAM with a learning rate of $10^{-3}$ for a total of 1000 iterations with a mini-batch size of one image to learn the parameters of this network.

Figure 9 shows the average per-pixel loss, the average pixel accuracy as well as the time per ADAM iteration vs. number of iterations used to solve the lower-level problem (inner iterations). This experiment shows that by solving the lower-level problem to higher accuracy the overall capacity and thus the accuracy of the system can be enhanced. This comes at the price of a higher computational complexity, which increases linearly with the number of iterations.

Finally, we perform a large scale experiment on this dataset. We partition the images into a training set of 572 images and use the remaining 143 images for testing. We use the pre-trained Fully Convolutional Network *FCN-32s* [25] as basis for this experiment. We adapt the geometry of the last two layers to this dataset and retrain the network. We then add a multiplicative scaling layer followed by the global segmentation model and refine the parameters for this network. The number of inner iterations was set to 100, since this setting provides a good tradeoff between accuracy and computational complexity. We use a mini-batch size of 5 images and a learning rate of $10^{-3}$.

The average accuracy in terms of the average pixel accuracy ($Acc$) in percent and Intersection over Union ($IoU$) score on both the test and the training set is shown in Table 1. We compare the plain Fully Convolutional Network *FCN* to the network with the additional global segmentation model *FCN+Global*. We observe a moderate increase of 1.4% in terms of IoU on the test

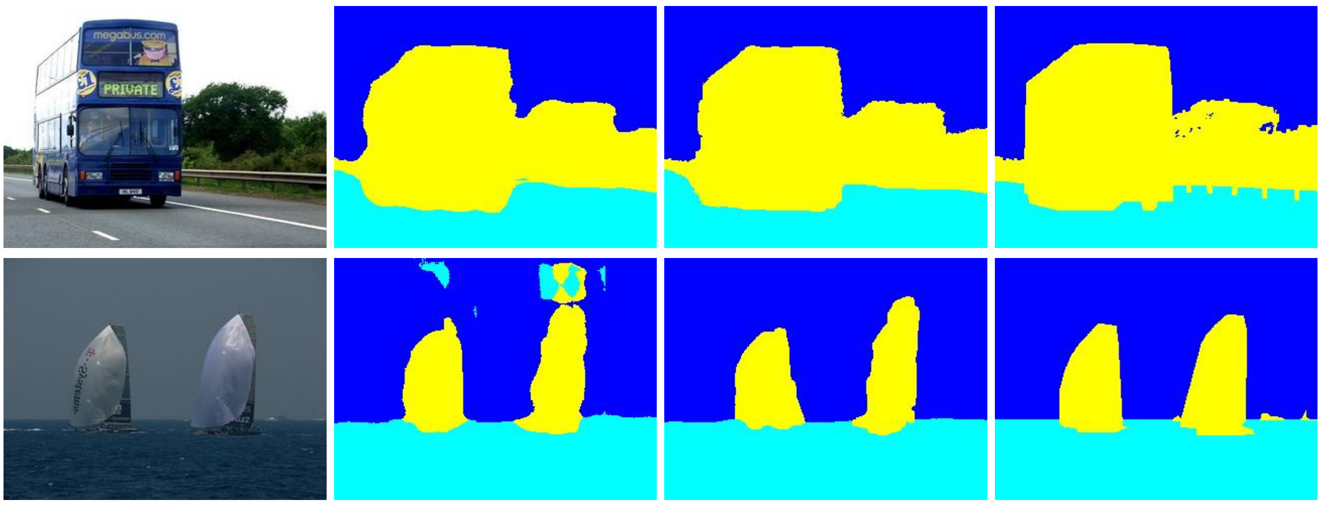


**Fig. 10** Example results from the test set. Row-wise, from left to right: Input image, CNN, CNN+Global, ground truth. The global model is able to align results to edges and is able to correct spurious errors.

set when using the global model. This can be attributed to the fact that the CNN alone already provides good but coarse segmentations and the segmentation model uses only simple pairwise interactions. As such it is unable to correct gross errors of the CNN.

Since the presented approach is applicable to a broad range of energies, training of more expressive energies which include more complex interactions (cf. [40]) is a promising direction of future research. Example segmentations from the test set are shown in Figure 10.

*Remark 5* The advantage of our approach versus the smoothing approach from Section 5.1 was already performed in the conference version of this paper. Instead of repeating the experiment, we refer to [29] for the experiment.

## 10 Conclusion

We considered a class of bilevel optimization problems with non-smooth lower level problem. By an appropriate approximation, for some problems, we can formulate an algorithm with a smooth update mapping that solves a non-smooth optimization problem in the lower level. This allows us to apply gradient based methods for solving the bilevel optimization problem. A second approach directly considers the fixed-point equation of the algorithm as optimality condition for the lower level problem. Key for both ideas are Bregman proximity functions.

The idea of estimating gradients for an abstract algorithm is exemplified for a forward–backward splitting method and a primal–dual algorithm with Bregman proximity functions. Several potential application examples are shown. Finally, a toy example confirms our results and provides some more intuition. The contribution of our idea to practical applications is demonstrated by a multi-label segmentation model that is coupled with a convolutional neural network.

Nevertheless, several open questions remain, which are highlighted in the manuscript. These questions need to be addressed in future work.

| | Test | | Train | |
|---|---|---|---|---|
| | Acc | IoU | Acc | IoU |
| FCN | 92.40 | 82.65 | 97.54 | 92.21 |
| FCN+Global | **93.00** | **84.01** | **97.90** | **93.53** |

**Table 1** Accuracy on the Stanford Background dataset [19]. We compare the plain CNN to the CNN with an additional global segmentation model.

Acknowledgment

Peter Ochs and Thomas Brox acknowledge support from the German Research Foundation (DFG grant BR 3815/8-1). René Ranftl acknowledges support from Intel Labs. Thomas Pock acknowledges support from the Austrian science fund under the ANR-FWF project "Efficient algorithms for nonsmooth optimization in imaging", No. I1148 and the FWF-START project "Bilevel optimization for Computer Vision", No. Y729.

## References

1. Al-Baali, M.: Descent Property and Global Convergence of the Fletcher–Reeves Method with Inexact Line Search. IMA Journal of Numerical Analysis **5**(1), 121–124 (1985)

2. Beck, A., Teboulle, M.: Mirror descent and nonlinear projected subgradient methods for convex optimization. Operations Research Letters **31**(3), 167–175 (2003)
3. Bennett, K., Kunapuli, G., Hu, J., Pang, J.S.: Bilevel Optimization and Machine Learning. In: J.M. Zurada, G.G. Yen, J. Wang (eds.) Computational Intelligence: Research Frontiers, no. 5050 in Lecture Notes in Computer Science, pp. 25–47. Springer Berlin Heidelberg (2008)
4. Bregman, L.M.: The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Computational Mathematics and Mathematical Physics **7**(3), 200–217 (1967)
5. Calatroni, L., Reyes, J., Schönlieb, C.B.: Dynamic sampling schemes for optimal noise learning under multiple nonsmooth constraints. ArXiv e-prints (2014). ArXiv: 1403.1278
6. Calatroni, L., Reyes, J., Schönlieb, C.B., Valkonen, T.: Bilevel approaches for learning of variational imaging models. ArXiv e-prints (2015). ArXiv: 1505.02120
7. Chambolle, A., Pock, T.: A first-order primal-dual algorithm for convex problems with applications to imaging. Journal of Mathematical Imaging and Vision **40**(1), 120–145 (2011)
8. Chambolle, A., Pock, T.: On the ergodic convergence rates of a first-order primal-dual algorithm. Tech. rep. (2014). To appear
9. Chen, Y., Pock, T., Ranftl, R., Bischof, H.: Revisiting Loss-Specific Training of Filter-Based MRFs for Image Restoration. In: J. Weickert, M. Hein, B. Schiele (eds.) German Conference on Pattern Recognition (GCPR), no. 8142 in Lecture Notes in Computer Science, pp. 271–281. Springer Berlin Heidelberg (2013)
10. Chen, Y., Ranftl, R., Pock, T.: Insights into analysis operator learning: From patch-based sparse models to higher order MRFs. IEEE Transactions on Image Processing **23**(3), 1060–1072 (2014)
11. Deledalle, C.A., Vaiter, S., Fadili, J., Peyré, G.: Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection. SIAM Journal on Imaging Sciences **7**(4), 2448–2487 (2014)
12. Dempe, S.: Annotated Bibliography on Bilevel Programming and Mathematical Programs with Equilibrium Constraints. Optimization **52**(3), 333–359 (2003)
13. Dempe, S., Kalashnikov, V., Pérez-Valdés, G., Kalashnykova, N.: Bilevel Programming Problems. Energy Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
14. Dempe, S., Zemkoho, A.: The Generalized Mangasarian–Fromowitz Constraint Qualification and Optimality Conditions for Bilevel Programs. Journal of Optimization Theory and Applications **148**(1), 46–68 (2010)
15. Domke, J.: Implicit Differentiation by Perturbation. In: Advances in Neural Information Processing Systems (NIPS), pp. 523–531 (2010)
16. Domke, J.: Generic methods for optimization-based modeling. In: International Workshop on Artificial Intelligence and Statistics, pp. 318–326 (2012)
17. Evans, L.C., Gariepy, R.F.: Measure Theory and Fine Properties of Functions. CRC Press, Boca Raton (1992)
18. Fletcher, R., Reeves, C.: Function minimization by conjugate gradients. The Computer Journal **7**(2), 149–154 (1964)
19. Gould, S., Fulton, R., Koller, D.: Decomposing a scene into geometric and semantically consistent regions. In: International Conference on Computer Vision (ICCV) (2009)
20. Griewank, A., Walther, A.: Evaluating Derivatives, second edn. Society for Industrial and Applied Mathematics (2008)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
22. Kunisch, K., Pock, T.: A bilevel optimization approach for parameter learning in variational models. SIAM Journal on Imaging Sciences **6**(2), 938–983 (2013)
23. Lions, P.L., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM Journal on Applied Mathematics **16**(6), 964–979 (1979)
24. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical Programming **45**(1), 503–528 (1989)
25. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
26. Moore, G.: Bilevel programming algorithms for machine learning model selection. Ph.D. thesis, Rensselaer Polytechnic Institute (2010)
27. Ochs, P.: Long term motion analysis for object level grouping and nonsmooth optimization methods. Ph.D. thesis, Albert–Ludwigs–Universität Freiburg (2015). URL `http://lmb.informatik.uni-freiburg.de//Publications/2015/Och15`
28. Ochs, P., Chen, Y., Brox, T., Pock, T.: ipiano: Inertial proximal algorithm for non-convex optimization. SIAM Journal on Imaging Sciences **7**(2), 1388–1419 (2014)
29. Ochs, P., Ranftl, R., Brox, T., Pock, T.: Bilevel optimization with nonsmooth lower level problems. In: International Conference on Scale Space and Variational Methods in Computer Vision (SSVM) (2015)
30. Passty, G.B.: Ergodic convergence to a zero of the sum of monotone operators in hilbert space. Journal of Mathematical Analysis and Applications **72**(2), 383 – 390 (1979)
31. Peyré, G., Fadili, J.: Learning analysis sparsity priors. In: Proceedings of Sampta (2011)
32. Ranftl, R., Pock, T.: A deep variational model for image segmentation. In: German Conference on Pattern Recognition (GCPR), pp. 107–118 (2014)
33. Reyes, J., Schönlieb, C.B., Valkonen, T.: The structure of optimal parameters for image restoration problems. ArXiv e-prints (2015). ArXiv: 1505.01953
34. Reyes, J.C.D.L., Schönlieb, C.B.: Image denoising: Learning noise distribution via pde-constrained optimisation. Inverse Problems and Imaging **7**, 1183–1214 (2013)
35. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (1970)
36. Tappen, M.: Utilizing variational optimization to learn MRFs. In: International Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–8 (2007)
37. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6**, 1453–1484 (2005)
38. Vedaldi, A., Lenc, K.: Matconvnet – convolutional neural networks for matlab (2015)
39. Zavriev, S., Kostyuk, F.: Heavy-ball method in nonconvex optimization problems. Computational Mathematics and Modeling **4**(4), 336–341 (1993)
40. Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., Torr, P.: Conditional random fields as recurrent neural networks. In: International Conference on Computer Vision (ICCV) (2015)